Xiao, J. and Volz, R.A. (1988). Design and motion constraints of part-mating planning in the presence of uncertainties. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1260–1268.

Xiao, J. and Volz, R.A. (1989). On replanning for assembly tasks using robots in the presence of uncertainties. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 638–645.

Yin, B., Ambler, A.P., and Popplestone, R.J. (1984). Combining vision verification with a high level robot programming language. In *Proceedings of the 4th International Conference, ROVISEC*, pages 371–379.

Srinivas, S. (1976). *Error Recovery in Robot Systems.* PhD thesis, California Institute of Technology.

Srinivas, S. (1978). Error recovery in robots through failure reason analysis. In *Proceedings of AFIPS National Computer Conference*, pages 275–282.

Takase, K., Paul, R.P., and Berg, E.J. (1981). A structured approach to robot programming and teaching. *IEEE Transactions on Systems Man. and Cybernatics*, SMC-11(4):274–239.

Tate, A. (1984). Planning and condition monitoring in an FMS. In *Proceedings of the International Conference on Flexible Automation Systems.*

Taylor, R.H. (1976). *The Synthesis of Manipulator Control Programs from Task-Level Specifications.* Report AIM 282, Stanford University.

Trevelyan, J.P., Nelson, M., and Kovesi, P. (1987). Adaptive motion sequencing for process robots. In *Proceedings of 4th International Symposium of Robotic Research*, pages 65–72.

Udupa, S.M. (1977). *Collision Detection and Avoidance in Computer Controlled Manipulators.* PhD thesis, California Institute of Technology.

Whitney, D.E. and Junkel, E.F. (1982). Applying stochastic control theory to robot sensing, teaching, and long term control. In *Proceedings of IFAC Conf. on Information Control Problems in Manufacturing Technology*, pages 109–117.

Williams, D.J., Rogers, P., and Upton, D.M. (1986). Programming and recovery in cells for factory automation. *The International Journal of Advanced Manufacturing Technology*, 1:37–47.

Wood, B.O. and Fugelso, M.A. April 1983. MCL, the manufacturing control language. In *Proceedings of the 13th International Symposium on Industrial Robots*, pages 12/85–96, Chicago.

Salmon, J.C. (1989). *Implementation of a Generalised Patting Behaviour for the SOMASS System.* Master's thesis, Department of Artificial Intelligence, University of Edinburgh.

Schraft, R.D., Strommer, W.M., and Neugebauer, J.G. October 1992. A virtual reality testbed for robot applications. In *Proceedings of the 23th International Symposium on Industrial Robots*, pages 105–110, Barcelona.

Selke, K.K.W., Shen, H.C., Deacon, G.E., and Pugh, A. (1991). A strategy for sensors and rules in flexible robotic assembly. *International Journal of Production Res.*, 29(2):279–291.

Sharp, K.A.I. (1990). *Measurement Generics For The Mk1 Infact Machine.* Design Document, Department of Computer Science, University College of Wales.

Shen, H.C., Selke, K.K.W., Deacon, G.E., and Pugh, A. (1987). A sensor-data driven, rule based strategy for error recovery in robotic assembly. In *Proceedings of NATO Advanced Research Workshop on Kinematic and Dynamic Issues in Sensor based Control.*

Shimano, B. (1979). VAL: a versatile robot programming and control system. In *Proceedings of COMPSAC.*

Shimano, B., Geschke, C., and Spalding, C. (1984). VAL-II: a robot programming language and control system. In *SME Robots VIII Conference.*

SICStus Prolog User's Manual. 0.6 patch level 7 edition, October 1988.

Smithers, T. and Malcolm, C.A. (1989). A behavioural approach to robot task planning and off–line programming. *Journal of Structured Learning*, 10:137–156.

Smithers, T., Malcolm, C.A., and Ambler, A.P. (1987). *Towards An Integrated Intelligent Robotic Assembly System.* DAI Working Paper 197, Department of Artificial Intelligence, University of Edinburgh.

Malcolm, C.A. and Smithers, T. (1990). Symbol grounding via a hybrid architecture in an autonomous assembly system. *Robotics and Autonomous Systems*, 6:123–144.

Malcolm, C.A. (1987). *Planning and Performing the Robotic Assembly of Soma Cube Constructions.* Master's thesis, Department of Artificial Intelligence, University of Edinburgh.

Malcolm, C.A., Smithers, T., and Hallam, J. (1989). *An Emerging Paradigm in Robotic Architecture.* DAI Research Paper 447, Department of Artificial Intelligence, University of Edinburgh.

Mani, Murali, and Wilson. (1985). A programmable orienting system for flat parts. In *Proceedings of NAMRII.*

Nevins, J.L., Desai, M., Fogel, E., Walker, B.K., and Whitney, D.E. (1984). Adaptive control, learning, and cost effective sensor systems for robotic or advanced automation systems. In *The First International Symposium on Robotics Research*, pages 983–994.

Peshkin, M.A. and Sanderson, A.C. December 1988. The motion of a pushed, sliding workpiece. *IEEE Journal of Robotics and Automation*, 4(6).

Peshkin, M.A. August 1990. Programmed compliance for error corrective assembly. *IEEE Transactions on Robotics and Automation*, 6(4).

Popplestone, R.J., Ambler, A.P., and Bellos, I. (1978). RAPT, a language for describing assemblies. *Industrial Robot*, 3:131–137.

Requicha. (1983). *Towards a Theory of Geometrical Tolerancing.* Tech. Memo 40, University of Rochester.

Introducing RTX. UMI Ltd, London, England, 3rd edition, April 1987.

Lozano-Pérez, T. (1981). Automatic planning of manipulator transfer movements. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC–11(10):681–698.

Lozano-Pérez, T. February 1983. Spatial planning : a configuration space approach. *IEEE Transactions on Computers*, C–32(2):108–120.

Lozano-Pérez, T., Jones, J.L., Mazer, E., O'Donnell, P.A., and Grimson, W.E.L. (1987). Handey: a robot system that recognises, plans, and manipulates. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 843–849.

Lozano-Pérez, T., Mason, M.T., and Taylor, R.H. (1990). *Artificial Intelligence at MIT : Expanding Frontiers*, chapter 27, pages 60–99. Volume 2, MIT Press, Automatic Synthesis of Fine-Motion Strategies for Robots.

Lozano-Pérez, T., Mazer, E., Jones, J.L., and O'Donnell, P.A. (1990). *Artificial Intelligence at MIT : Expanding Frontiers*, chapter 26, pages 40–59. Volume 2, MIT Press, Task Level Planning of Pick-and-Place Robot Motions.

Malcolm, C.A. and Fothergill, A.P. (1986). *Some Architectural Implications of the use of Sensors.* DAI Research Paper 294, Department of Artificial Intelligence, University of Edinburgh.

Malcolm, C.A. and Howe, J. (1990). *Behavioural Modules: A New Approach to Robotic Assembly.* ACME Grant GR/E 68075 - Extended Report, Department of Artificial Intelligence, University of Edinburgh.

Malcolm, C.A. and Smithers, T. (1988). *Programming Assembly Robots in terms of Tack Achieving Behavioural Modules: First Experimental Results.* DAI Research Paper 410, Department of Artificial Intelligence, University of Edinburgh.

Jennings, J., Donald, B.R., and Campbell, D. (1989). Towards experimental verification of an automated compliant motion planner based on a geometric theory of error detection and recovery. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 632–637.

Johnson, D.G and Hill, J.J. September 1985. A kalman filter approach to sensor-based robot control. *IEEE Journal of Robotics and Automation*, RA-1(3).

Kalman, R.E. March 1960. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82D:35–45.

Lee, M.H., Barnes, D.P., and Hardy, N.W. (1983). Knowledge based error recovery in industrial robots. In *Proceedings of 8th International Joint Conference on Artificial Intelligence*, pages 824–826.

Lee, M.H., Hardy, N.W., and Barnes, D.P. (1984). Research into automatic error recovery. In *Proceedings of Conference on UK Robotics Research*, pages 65–69.

Lieberman, L.I. and Wesley, M.A. (1977). Autopass: an automatic programming system for computer controlled mechanical assembly. *IBM Journal of Research and Development*, 21(4):321–333.

Loughlin, C., (ed.). (1992). *InFACT: Project, Concepts, Machine.* MCB University Press Limited, Bradford.

Lozano-Pérez, T. and Brooks, R.A. April 1985. *An Approach to Automatic Robot Programming.* AI Memo 842, MIT.

Lozano-Pérez, T. and Wesley, M.A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570.

Lozano-Pérez, T. (1976). *The Design of a Mechanical Assembly System.* Technical Report 397, Artificial Intelligence laboratory, MIT.

Fleming, A. (1985). *Analysis of Uncertainty in a Structure of Parts 1.* DAI Research Paper 271, Department of Artificial Intelligence, University of Edinburgh.

Fleming, A. (1985). *Analysis of Uncertainty in a Structure of Parts 2.* DAI Research Paper 272, Department of Artificial Intelligence, University of Edinburgh.

Fleming, A.D. (1987). *Analysis of Uncertainties and Geometric Tolerances in Assemblies of Parts.* PhD thesis, Department of Artificial Intelligence, University of Edinburgh.

Flynn, A.M. and Brooks, R.A. (1988). MIT robots - what's next? In *Proceedings of IEEE Robotics and Automation Conference*, pages 611–617.

Gardner, M. (1961). *More Mathemetical Puzzles and Diversions.* Penguin, 1966 edition.

Gini, M. and Gini, G. (1983). Recovering from failures : a new challenge for industrial robots. In *Proceedings of 26th IEEE Computer Society Intern. Conf.*, pages 220–227.

Gini, M. and Gini, G. (1983). Towards automatic error recovery in robot programs. In *Proceedings of 8th International Joint Conference on Artificial Intelligence*, pages 821–823.

Gini, M. and Smith, R.E. (1986). Reliable real–time robot operation employing intelligent forward recovery. *Journal of Robotic Systems*, 3:281–300.

Hardy, N.W. Robotic Assembly Workshop Presentation, Edinburgh University, (1991).

Hardy, N.W., Barnes, D.P., and Lee, M.H. (1989). The automatic diagnosis of task faults in flexible manufacturing systems. *Robotica*, 7(1):25–35.

Craig, J.J. (1989). *Introduction to Robotics, Mechanics and Control. 2nd Edition*, pages 1–144. Addison Wesley.

Critchlow, A.J. (1985). *Introduction to Robotics*, chapter 8, pages 293–324. MacMillan.

Dean, G.C. March 1986. An introduction to kalman filters. *Measurement + Control, The Institute of Measurement and Control*, 19(2).

Derby, S. March 1982. Computer graphics simulation of robot arms using the GRASP program. In *Proceedings of CAD/CAM conference*, MIT, Cambridge.

Donald, B.R. April 1986. Robot motion planning with uncertainty in the geometric models of the robot and environment : a formal framework for error detection and recovery. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1588–1593, San Francisco.

Donald, B.R. (1987). *Error Detection and Recovery for Robot Motion Planning with Uncertainty.* PhD thesis, MIT Artificial Intelligence Laboratory.

Donald, B.R. (1988). Planning multi-step error detection and recovery strategies. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 892–897.

Donald, B.R. February 1990. Planning multi-step error detection and recovery strategies. *International Journal of Robotics Research*, 9(1):3–60.

Drabble, B. and Cox, P. (1985). *Error Detection and Recovery in an Uncertain Environment.* Technical Report, Department of Computer Science and Applied Mathematics, University of Aston, Birmingham.

Erdmann, M.A. and Mason, M.T. August 1988. An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, 4(4):369–379.

Erdmann, M.A. (1984). *On Motion Planning with Uncertainty.* Technical Report 810, MIT.

Brooks, R.A. (1987). A hardware retargetable distributed layered architecture for mobile robot control. In *Proceedings of IEEE Robotics and Automation Conference*, pages 106–110.

Brooks, R.A. (1989). A robot that walks: emergent behaviors from a carefully evolved network. In *Proceedings of IEEE Robotics and Automation Conference*, pages 692–696.

Brooks, R.A. (1991). *Intelligence Without Reason.* AI Memo 1293, MIT.

Brooks, R.A. (1991). Intelligence without representation. *Artificial Intelligence*, 47:139–160.

Brost, R. April 1986. Automatic grasp planning in the presence of uncertainty. In *Proceedings of IEEE International Conference on Robotics and Automation.*

Cameron, S. and Aylett, J. (1988). Robmod: a geometry engine for robotics. In *Proceedings of IEEE Conference on Robotics and Automation.*

Cameron, S.A. (1984). *Modelling Solids in Motion.* PhD thesis, Department of Artificial Intelligence, University of Edinburgh.

Chongstitvatana, P. and Conkie, A.D. (1990). *Behaviour-Based Assembly Experiments using Vision Sensing.* DAI Research Paper 466, Department of Artificial Intelligence, University of Edinburgh.

Clocksin, B. and Mellish, C. (1984). *Programming in Prolog.* Springer.

Connell, J.H. (1989). *A Colony Architecture for an Artificial Creature.* PhD thesis, Massachusetts Institute of Technology.

Corner, D.F., Ambler, A.P., and Popplestone, R.J. (1983). *Reasoning about the Spatial Relationships derived from a RAPT Program for describing assembly by Robot.* DAI Research Paper 191, Department of Artificial Intelligence, University of Edinburgh.

Bibliography

Adept manipulator system basic operation. 1212 Bordeaux Drive, Sunnyvale, CA, 1.5 edition, February 1985.

Adept manipulator system VAL–II reference guide. 1212 Bordeaux Drive, Sunnyvale, CA, 4.0 edition, July 1985.

Albus, J.S. (1981). *Brains, Behavior, and Robotics.* BYTE Publications Inc.

Ambler, A.P., Brown, C.M., Burstall, R.M., Popplestone, R.J., and Barrow, H.G. (1973). A versatile computer-controlled assembly system. In *Proceedings of 3rd International Joint Conference on AI*, pages 298–307.

Ambler, A.P., Cameron, S.A., and Corner, D.F. (1987). *Languages for Sensor-based control in Robotics*, pages 305–316. Volume F29, Springer-Verlag : NATO ASI Series, Augmenting the RAPT Robot Language.

Berlekamp, Conway, and Guy. (1982). *Winning Ways.* Volume 2, Academic Press.

Brooks, R.A. (1982). *Symbolic Error Analysis and Robot Planning.* AI Memo 685, MIT.

Brooks, R.A. (1986). *Achieving Artificial Intelligence Through Building Robots.* AI Memo 899, MIT, Artificial Intelligence Laboratory.

Brooks, R.A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2:14–23.

```
                  100          101 102          103
PLAN              Y            Y   Y            Y
set_up_init       Y            Y   Y            Y

PICK-UP           YYYYYYYYYYYY YYY YYYYYYYYYYYY YYYYYYY
FIND LOCATION     YYYYYYYYYYYY YYY YYYYYYYYYYYY YYYYYYY
FIND CUBE TOP     YYYYYYYYYYYY YYY YYYYYYYYYYYY YYYYYYY
FIND HEIGHT       YYYYYYYYYYYY YYY YYYYYYYYYYYY YYYYYYY
  FIND CENTRE
  HUNT CENTRE
  CHECKPRESSED
  FIND HEIGHT
  COMPARE HEIGHT
LOCATION          YYYYYYYYYYYY YYY YYYYYYYYYYYY YYYYYYY
FIND DROP         987969754787 458 874889467798 4878978
COMPARE POINTS    YYYYYYYYYYYY YYY YYYYYYYYYYYY YYYYYYY

PICK DOUBLE SNAP  Y    Y  YY   Y   Y    Y  YY   Y    Y
PICK UP           Y    Y  YY   Y   Y    Y  YY   Y    Y
  SINGLE PICK      YYYY YY  YY  YC  YYYY YY  YY  YYYY Y
  FIND HEIGHT      YYYY YY  YY  Y   YYYY YY  YY  YYYY Y
  FIND DROP        YYYY YY  YY  Y   YYYY YY  YY  YYYY Y
  PICK UP          YYYY YY  YY  Y   YYYY YY  YY  YYYY Y

  REGRASP                    Y                Y
    turn for put             Y                Y
  REGRASP PUT                Y                Y
    REGRASP TWIST            Y                Y
    REGRASP GET              M                M
    REGRASP PUT
  REGRASP GET

PLACE             YYYYYYYY     YY  YYYYYYYY     YYYYYYY
  SLIP                                                E
CHECK PUSH        YYYYYYYY     YY  YYYYYYYY     YYYYYY
  PUSH
  MOVE ROUND ASS          YYY              YYY
  approach ass            YYY              YYY
  place part              YYY              YYY
  depart ass              YYY              YYY
  CHECK PUSH              YYY              YYY
    approach ass
    push part
    depart ass
TIDY UP
```

	91	92	93	94	95	96	97	98	99
PLAN	Y	Y	Y	Y	Y	Y	Y	Y	Y
set_up_init	Y	Y	Y	Y	Y	Y	Y	Y	Y
PICK-UP	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND LOCATION	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND CUBE TOP	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND HEIGHT	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND CENTRE									
HUNT CENTRE									
CHECKPRESSED									
FIND HEIGHT									
COMPARE HEIGHT									
LOCATION	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND DROP	9456845	8444854	9944754	4554455	4545475	5645447	4949445	4858444	4947474
COMPARE POINTS	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
PICK DOUBLE SNAP	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY
PICK UP	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY
SINGLE PICK	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND HEIGHT	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND DROP	Y	Y	Y	Y	Y	Y	Y	Y	Y
PICK UP	Y	Y	Y	Y	Y	Y	Y	Y	Y
REGRASP	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y
turn for put									
REGRASP PUT	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y
REGRASP TWIST	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y
REGRASP GET	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y
REGRASP PUT	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y
REGRASP GET	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y
PLACE	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
SLIP									
CHECK PUSH	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
PUSH	Y	Y	Y	Y	Y	Y	Y	Y	Y
MOVE ROUND ASS									
approach ass									
place part									
depart ass									
CHECK PUSH									
approach ass									
push part									
depart ass									
TIDY UP	Y	Y	Y	Y	Y	Y	Y	Y	Y

	81	82	83	84	85	86	87	88	89	90
PLAN	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
set_up_init	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PICK-UP	YYYYYYY	YYYYYYY	YYYYYYY	Y	YYYYYYY	YYYYYYY	YYYYYYY	Y	YYYYY	YYYYYYY
FIND LOCATION	YYYYYYY	YYYYYYY	YYYYYYY	Y	YYYYYYY	YYYYYYY	YYYYYYY	Y	YYYYY	YYYYYYY
FIND CUBE TOP	YYYYYYY	YYYYYYY	YYYYYYY	Y	YYYYYYY	YYYYYYY	YYYYYYY	Y	YYYYY	YYYYYYY
FIND HEIGHT	YYYYYYY	YYYYYYY	YYYYYYY	Y	YYYYYYY	YYYYYYY	YYYYYYY	Y	YYYYY	YYYYYYY
FIND CENTRE										
HUNT CENTRE										
CHECKPRESSED										
FIND HEIGHT										
COMPARE HEIGHT										
LOCATION	YYYYYYY	YYYYYYY	YYYYYYY	Y	YYYYYYY	YYYYYYY	YYYYYYY	Y	YYYYY	YYYYYYY
FIND DROP	5755885	8458844	8459774	6	6845444	7448448	7444548	6	86484	12565774
COMPARE POINTS	YYYYYYY	YYYYYYY	YYYYYYY	Y	YYYYYYY	YYYYYYY	YYYYYYY	Y	YYYYY	YYYYYYY
PICK DOUBLE SNAP	YYY YYY	YYY YYY	YYY YYY	L	YYY YYY	YYY YYY	YYY YYY	C	YYY Y	YYY YYY
PICK UP	YYY YYY	YYY YYY	YYY YYY		YYY YYY	YYY YYY	YYY YYY		YYY Y	YYY YYY
SINGLE PICK	Y	Y	Y		Y	Y	Y		Y	Y
FIND HEIGHT	Y	Y	Y		Y	Y	Y		Y	Y
FIND DROP	Y	Y	Y		Y	Y	Y		Y	Y
PICK UP	Y	Y	Y		Y	Y	Y		Y	Y
REGRASP	Y Y	Y Y	Y Y		Y Y	Y Y	Y Y		Y	Y Y
turn for put										
REGRASP PUT	Y Y	Y Y	Y Y		Y Y	Y Y	Y Y		Y	Y Y
REGRASP TWIST	Y Y	Y Y	Y Y		Y Y	Y Y	Y Y		Y	Y Y
REGRASP GET	Y Y	Y Y	Y Y		Y Y	Y Y	Y Y		Y	Y Y
REGRASP PUT	Y Y	Y Y	Y Y		Y Y	Y Y	Y Y		Y	Y Y
REGRASP GET	Y Y	Y Y	Y Y		Y Y	Y Y	Y Y		F	Y Y
PLACE	YYYYYYY	YYYYYYY	YYYYYYY		YYYYYYY	YYYYYYY	YYYYYYY		YYYY	YYYYYYY
SLIP										
CHECK PUSH	YYYYYYY	YYYYYYY	YYYYYYY		YYYYYYY	YYYYYYY	YYYYYYY		YYYY	YYYYYYY
PUSH	Y	Y	Y		Y	Y	Y			Y
MOVE ROUND ASS										
approach ass										
place part										
depart ass										
CHECK PUSH										
approach ass										
push part										
depart ass										
TIDY UP	Y	Y	Y		Y	Y	Y			Y

	71	72	73	74	75	76	77	78	79	80
PLAN	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
set_up_init	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PICK-UP	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND LOCATION	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND CUBE TOP	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND HEIGHT	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND CENTRE										
HUNT CENTRE										
CHECKPRESSED										
FIND HEIGHT										
COMPARE HEIGHT										
LOCATION	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND DROP	8654684	8548474	945K	9846545	4847744	44577	8855964	4949754	9487754	4545565
COMPARE POINTS	YYYYYYY	YYYYYYY	YYY	YYYYYYY	YYYYYYY	YYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
PICK DOUBLE SNAP	YYY YYY	YYY YYY	YYY	YYY YYY	YYY YYY	YYY C	YYY YYY	YYY YYY	YYY YYY	YYY YYY
PICK UP	YYY YYY	YYY YYY	YYY	YYY YYY	YYY YYY	YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY
SINGLE PICK	Y	Y		Y	Y	Y	Y	Y	Y	Y
FIND HEIGHT	Y	Y		Y	Y	Y	Y	Y	Y	Y
FIND DROP	Y	Y		Y	Y	Y	Y	Y	Y	Y
PICK UP	Y	Y		Y	Y	Y	Y	Y	Y	Y
REGRASP	Y YYY	Y YYY	Y	Y YYY	Y YYY	Y	Y YYY	Y YYY	Y YYY	Y Y
turn for put										
REGRASP PUT	Y YYY	Y YYY	Y	Y YYY	Y YYY	Y	Y YYY	Y YYY	Y YYY	Y Y
REGRASP TWIST	Y Y	Y Y		Y Y	Y Y		Y Y	Y Y	Y Y	Y Y
REGRASP GET	Y Y	Y Y		Y Y	Y Y		Y Y	Y Y	Y Y	Y Y
REGRASP PUT	Y Y	Y Y		Y Y	Y Y		Y Y	Y Y	Y Y	Y Y
REGRASP GET	Y YYY	Y YYY	Y	Y YYY	Y YYY	Y	Y YYY	Y YYY	Y YYY	Y Y
PLACE	YYYYYYY	YYYYYYY	YYY	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
SLIP										
CHECK PUSH	YYYYYYY	YYYYYYY	YYY	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
PUSH	Y	Y		Y	Y		Y	Y	Y	Y
MOVE ROUND ASS										
approach ass										
place part										
depart ass										
CHECK PUSH										
approach ass										
push part										
depart ass										
TIDY UP	Y	Y		Y	Y		Y	Y	Y	Y

	61	62	63	64	65	66	67	68	69	70
PLAN	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
set_up_init	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PICK-UP	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND LOCATION	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND CUBE TOP	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND HEIGHT	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND CENTRE							Y			
HUNT CENTRE							1			
CHECKPRESSED							1			
FIND HEIGHT							1			
COMPARE HEIGHT							1			
LOCATION	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND DROP	13944844	9455684	7444474	5484644	4474456	8457454	4645474	5465744	9858644	4848744
COMPARE POINTS	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
PICK DOUBLE SNAP	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY
PICK UP	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY	YYY YYY
SINGLE PICK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND HEIGHT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND DROP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PICK UP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
REGRASP	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY
turn for put										
REGRASP PUT	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY
REGRASP TWIST	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y
REGRASP GET	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y
REGRASP PUT	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y
REGRASP GET	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY	Y YYY
PLACE	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
SLIP										
CHECK PUSH	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
PUSH	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
MOVE ROUND ASS										
approach ass										
place part										
depart ass										
CHECK PUSH										
approach ass										
push part										
depart ass										
TIDY UP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

	51	52	53	54	55	56	57	58	59	60
PLAN	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
set_up_init	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PICK-UP	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND LOCATION	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND CUBE TOP	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND HEIGHT	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND CENTRE										
HUNT CENTRE										
CHECKPRESSED										
FIND HEIGHT										
COMPARE HEIGHT										
LOCATION	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND DROP	4474454	8765575	4875445	4774455	4867477	8475488	4744584	4744444	4476744	7844444
COMPARE POINTS	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
PICK DOUBLE SNAP	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYY YYY
PICK UP	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYY YYY
SINGLE PICK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND HEIGHT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND DROP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PICK UP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
REGRASP	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y YYY
turn for put										
REGRASP PUT	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y YYY
REGRASP TWIST										Y Y
REGRASP GET										Y Y
REGRASP PUT										Y Y
REGRASP GET	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y YYY
PLACE	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYJ	YYYYYYY	YYYYYYY	YYYYYYY
SLIP										
CHECK PUSH	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
PUSH	Y	Y	Y	Y	Y	Y		Y	Y	Y
MOVE ROUND ASS										
approach ass										
place part										
depart ass										
CHECK PUSH										
approach ass										
push part										
depart ass										
TIDY UP	Y	Y	Y	Y	Y	Y		Y	Y	Y

	41	42	43	44	45	46	47	48	49	50
PLAN	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
set_up_init	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PICK-UP	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND LOCATION	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND CUBE TOP	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND HEIGHT	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND CENTRE								Y		
HUNT CENTRE								3		
CHECKPRESSED								3		
FIND HEIGHT								3		
COMPARE HEIGHT								1		
LOCATION	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND DROP	5766465	4874775	4766	4844558	4974898	8748494	4886454	8464477	4874485	4465444
COMPARE POINTS	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
PICK DOUBLE SNAP	YYYYY Y	YYYYY Y	YYYY	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y
PICK UP	YYYYY Y	YYYYY Y	YYYY	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y	YYYYY Y
SINGLE PICK	Y	Y		Y	Y	Y	Y	Y	Y	Y
FIND HEIGHT	Y	Y		Y	Y	Y	Y	Y	Y	Y
FIND DROP	Y	Y		Y	Y	Y	Y	Y	Y	Y
PICK UP	Y	Y		Y	Y	Y	Y	Y	Y	Y
REGRASP	Y Y	Y Y		Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y
turn for put										
REGRASP PUT	Y Y	Y Y		Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y
REGRASP TWIST										
REGRASP GET										
REGRASP PUT										
REGRASP GET	Y Y	Y Y		Y Y	Y Y	Y Y	Y Y	Y Y	Y Y	Y Y
PLACE	YYYYYYY	YYYYYYY	YYYI	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
SLIP										
CHECK PUSH	YYYYYYY	YYYYYYY	YYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
PUSH	Y	Y		Y	Y	Y	Y	Y	Y	Y
MOVE ROUND ASS										
approach ass										
place part										
depart ass										
CHECK PUSH										
approach ass										
push part										
depart ass										
TIDY UP	Y	Y		Y	Y	Y	Y	Y	Y	Y

	31	32	33	34	35	36	37	38	39	40
PLAN	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
set_up_init	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PICK-UP	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND LOCATION	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND CUBE TOP	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND HEIGHT	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND CENTRE										
HUNT CENTRE										
CHECKPRESSED										
FIND HEIGHT										
COMPARE HEIGHT										
LOCATION	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND DROP	4784458	5485484	7446587	5664468	8744465	6765457	894758	5454487	4744584	7444575
COMPARE POINTS	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
PICK DOUBLE SNAP	YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY
PICK UP	YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY	YYYYYY
SINGLE PICK	Y	Y	Y	Y	Y	Y		Y	Y	Y
FIND HEIGHT	Y	Y	Y	Y	Y	Y		Y	Y	Y
FIND DROP	Y	Y	Y	Y	Y	Y		Y	Y	Y
PICK UP	Y	Y	Y	Y	Y	Y		Y	Y	Y
REGRASP	Y	Y	Y	Y	Y	Y		Y	Y	Y
turn for put										
REGRASP PUT	Y	Y	Y	Y	Y	Y		Y	Y	Y
REGRASP TWIST										
REGRASP GET										
REGRASP PUT										
REGRASP GET	Y	Y	Y	Y	Y	Y		Y	Y	Y
PLACE	YYYYY	YYYYY	YYYYY	YYYYY	YYYYY	YYYYY	YYYYY	YYYYY	YYYYY	YYYYY
SLIP										
CHECK PUSH	YYYYY	YYYYY	YYYYY	YYYYY	YYYYY	YYYYY	YYYYY	YYYYY	YYYYY	YYYYY
PUSH										
MOVE ROUND ASS	YY	YY	YY	YY	YY	YY	Y	YY	YY	YY
approach ass	YY	YY	YY	YY	YG	YY	H	YY	YY	YY
place part	YY	YY	YY	YY	Y	YY		YY	YY	YY
depart ass	YY	YY	YY	YY	Y	YY		YY	YY	YY
CHECK PUSH	YY	YY	YY	YY	Y	YY		YY	YY	YY
approach ass										
push part										
depart ass										
TIDY UP	Y	Y	Y	Y		Y		Y	Y	Y

	21	22	23	24	25	26	27	28	29	30
PLAN	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
set_up_init	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PICK-UP	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYB	YYYYYYY	YYYYYYY	YYYYYYY
FIND LOCATION	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND CUBE TOP	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND HEIGHT	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND CENTRE	Y	Y	YY							
HUNT CENTRE	3	2	11							
CHECKPRESSED	3	2	11							
FIND HEIGHT	3	2	11							
COMPARE HEIGHT	3	1	11							
LOCATION	YYYYYYY	YYYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND DROP	7764584	7757449	546A	8848465	7844444	7544585	846	4747784	7849488	8794445
COMPARE POINTS	YYYYYYY	YYYYYYY	YYY	YYYYYYY	YYYYYYY	YYYYYYY	YYY	YYYYYYY	YYYYYYY	YYYYYYY
PICK DOUBLE SNAP	YYYYYY	YYYYYY	YYY	YYYYYY	YYYYYY	YYYYYY	YYY	YYYYYY	YYYYYY	YYYYYY
PICK UP	YYYYYY	YYYYYY	YYY	YYYYYY	YYYYYY	YYYYYY	YYY	YYYYYY	YYYYYY	YYYYYY
SINGLE PICK	Y	Y		Y	Y	Y		Y	Y	Y
FIND HEIGHT	Y	Y		Y	Y	Y		Y	Y	Y
FIND DROP	Y	Y		Y	Y	Y		Y	Y	Y
PICK UP	Y	Y		Y	Y	Y		Y	Y	Y
REGRASP	Y	Y		Y	Y	Y		Y	Y	Y
turn for put										
REGRASP PUT	Y	Y		Y	Y	Y		Y	Y	Y
REGRASP TWIST										
REGRASP GET										
REGRASP PUT										
REGRASP GET	Y	Y		Y	Y	Y		Y	Y	Y
PLACE	YYYYY	YYYYY	YYY	YYYYY	YYYYY	YYYYY	YYY	YYYYY	YYYYY	YYYYY
SLIP										
CHECK PUSH	YYYYY	YYYYY	YYY	YYYYY	YYYYY	YYYYY	YYY	YYYYY	YYYYY	YYYYY
PUSH										
MOVE ROUND ASS	YY	YY		YY	YY	YY		YY	YY	YY
approach ass	YY	YY		YY	YY	YY		YY	YY	YY
place part	YY	YY		YY	YY	YY		YY	YY	YY
depart ass	YY	YY		YY	YY	YY		YY	YY	YY
CHECK PUSH	YY	YY		YY	YY	YY		YY	YY	YY
approach ass										
push part										
depart ass										
TIDY UP	Y	Y		Y	Y	Y		Y	Y	Y

```
                  11  12      13      14    15      16    17 18   19      20

PLAN              Y   Y       Y       Y     Y       Y     Y  Y    Y       Y

set_up_init       Y   Y       Y       Y     Y       Y     Y  Y    Y       Y


PICK-UP           YYY YYYYYYY YYYYYYY YYYYY YYYYYYY YYYYY YY YYYY YYYYYYY YYYYYYY

FIND LOCATION     YYY YYYYYYY YYYYYYY YYYYY YYYYYYY YYYYY YY YYYY YYYYYYY YYYYYYY

FIND CUBE TOP     YYY YYYYYYY YYYYYYY YYYYY YYYYYYY YYYYY YY YYYY YYYYYYY YYYYYYY

FIND HEIGHT       YYY YYYYYYY YYYYYYY YYYYY YYYYYYY YYYYY YB YYYY YYYYYYY YYYYYYY

  FIND CENTRE                                                           Y   Y

  HUNT CENTRE                                                           1   1

  CHECKPRESSED                                                          1   1

  FIND HEIGHT                                                           1   1

  COMPARE HEIGHT                                                        1   1

LOCATION          YYY YYYYYYY YYYYYYY YYYYY YYYYYYY YYYYY Y  YYYY YYYYYYY YYYYYYY

FIND DROP         95D 4665467 7874497 77685 6844584 88775 4  6784 9444445 7674475

COMPARE POINTS    YY  YYYYYYY YYYYYYY YYYYY YYYYYYY YYYYY Y  YYYY YYYYYYY YYYYYYY


PICK DOUBLE SNAP  YY  YY YYYY YYYYY Y YYYYY YYYYY Y YYYYY Y  YYY  YYYYYY  YYYYYY

PICK UP           YY  YY YYYY YYYYY Y YYYYY YYYYY Y YYYYY Y  YYY  YYYYYY  YYYYYY

  SINGLE PICK           Y          Y             Y              Y       Y       Y

  FIND HEIGHT           Y          Y             Y              Y       Y       Y

  FIND DROP             Y          Y             Y              Y       Y       Y

  PICK UP               Y          Y             Y              Y       Y       Y


  REGRASP         Y   Y   YYY     Y Y     Y     Y Y     Y Y  Y YY       Y       Y

    turn for put

  REGRASP PUT     Y   Y   YYY     Y Y     Y     Y Y     Y Y  Y YY       Y       Y

    REGRASP TWIST         Y Y     Y       Y     Y       Y Y  Y Y

    REGRASP GET           Y Y     Y       Y     Y       Y Y  Y Y

    REGRASP PUT           Y Y     Y       Y     Y       Y Y  Y Y

  REGRASP GET     Y   Y   Y Y     Y Y     Y     Y Y     Y Y  Y YF       Y       Y


PLACE             YY  YYYYYYY YYYYYYY YYYYY YYYYYYY YYYYY Y  YYY  YYYYY   YYYYY

 SLIP                       E             E             E

CHECK PUSH        YY  YYYYYY  YYYYYYY       YYYYYYY       Y  YYY  YYYYY   YYYYY

  PUSH                              Y             Y                     Y       Y

  MOVE ROUND ASS                                                       YY      YY

  approach ass                                                         YY      YY

  place part                                                           YY      YY

  depart ass                                                           YY      YY

  CHECK PUSH                                                           YY      YY

    approach ass

    push part

    depart ass

TIDY UP                             Y             Y                     Y       Y
```

	1	2	3	4	5	6	7	8	9	10
PLAN	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
set_up_init	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PICK-UP	YYYYYYY	YYYYYYY	YYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND LOCATION	YYYYYYY	YYYYYYY	YYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND CUBE TOP	YYYYYYY	YYYYYYY	YYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND HEIGHT	YYYYYYY	YYYYYYY	YYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND CENTRE						Y	Y			
HUNT CENTRE						1	4			
CHECKPRESSED						1	4			
FIND HEIGHT						1	4			
COMPARE HEIGHT						1	2			
LOCATION	YYYYYYY	YYYYYYY	YYYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
FIND DROP	4447469	4855566	44546A	8744	9444599	4847774	4744487	7444769	7444747	4946477
COMPARE POINTS	YYYYYYY	YYYYYYY	YYYYY	YYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY
PICK DOUBLE SNAP	YY YYYY	YY YYYY	YY YY	YY Y	YY YYYY	YY YYYY	YY YYYY	YY YYYY	YY YYYC	YY YYYY
PICK UP	YY YYYY	YY YYYY	YY YY	YY Y	YY YYYY	YY YYYY	YY YYYY	YY YYYY	YY YYY	YY YYYY
SINGLE PICK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND HEIGHT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND DROP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PICK UP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
REGRASP	Y YYY	Y YYY	Y Y	Y	Y YYY	Y YYY	Y YYY	Y YYY	Y YY	Y YYY
turn for put										
REGRASP PUT	Y YYY	Y YYY	Y Y	Y	Y YYY	Y YYY	Y YYY	Y YYY	Y YY	Y YYY
REGRASP TWIST	Y Y	Y Y	Y		Y Y	Y Y	Y Y	Y Y	Y	Y Y
REGRASP GET	Y Y	Y Y	Y		Y Y	Y Y	Y Y	Y Y	Y	Y Y
REGRASP PUT	Y Y	Y Y	Y		Y Y	Y Y	Y Y	Y Y	Y	Y Y
REGRASP GET	Y YYY	Y YYY	Y Y	Y	Y YYY	Y YYY	Y YYY	Y YYY	Y YY	Y YYY
PLACE	YYYYYYY	YYYYYYY	YYYYY	YYYB	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYY	YYYYYYY
SLIP										
CHECK PUSH	YYYYYYY	YYYYYYY	YYYYY	YYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYYY	YYYYYY	YYYYYYY
PUSH	Y	Y			Y	Y	Y	Y		Y
MOVE ROUND ASS										
approach ass										
place part										
depart ass										
CHECK PUSH										
approach ass										
push part										
depart ass										
TIDY UP	Y	Y			Y	Y	Y	Y		Y

Appendix F

Whole Run Tables

The following tables detail the runs of the experimental robotic assembly system described in Chapter 6. These runs involve the Behavioural Modules of the whole assembly system, although only the ones important to the thesis work are actually detailed here. The assembly runs are looked at in Section 6.2 of the above chapter.

The run numbers head the columns of the table, and the row headers show the Behavioural Module names which are used. The ones picked out in capital letters are the Behavioural Modules in upper level of the hierarchy, with the others forming the lowest level Behavioural Modules in the structure. The Behavioural Modules not on the *Ideal Execution Path* are indented. The numbers shown against the **Hunt Centre** and **Find Drop** Behavioural Modules are described in the explanation for the part acquisition Behavioural Modules runs in Appendix E. For these runs, each run consisted of initialisation and tidy up sections, with several runs of each pick/place section dependent on the number of Soma parts to be assembled. The diagrams also show where failures occurred, and the actual cause of the errors is detailed in Section 6.2.

	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
NORMAL PICK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND LOCATION	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND CUBE TOP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND HEIGHT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
safe height	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
move2/check2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
movesone	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
checkone	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
move																									
checkone																									
checkone																									
FIND CENTRE	Y																Y	Y	Y		Y	Y	Y		Y
HUNT CENTRE	1																1	1	1		1	1	1		1
CHECKPRESSED	1																1	1	1		1	1	1		1
FIND HEIGHT	1																1	1	1		1	1	1		1
safe height	1																1	1	1		1	1	1		1
move2/check2	1																1	1	1		1	1	1		1
movesone	1																1	1	1		1	1	1		1
checkone	1																1	1	1		1	1	1		1
move																									
checkone																									
checkone																									
COMPARE HEIGHT	1																1	1	1		1	1	1		1
LOCATION	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND DROP	9	8	4	7	5	8	5	4	12	9	4	5	5	4	5	5	8	9	9	5	8	9	4	4	9
appros2/check2	9	8	4	7	5	8	5	4	12	9	4	5	5	4	5	5	8	9	9	5	8	9	4	4	9
movesone	9	8	4	7	5	8	5	4	12	9	4	5	5	4	5	5	8	9	9	5	8	9	4	4	9
checkone	9	8	4	7	5	8	5	4	12	9	4	5	5	4	5	5	8	9	9	5	8	9	4	4	9
movesone	9	8	4	7	5	8	5	4	12	9	4	5	5	4	5	5	8	9	9	5	8	9	4	4	9
checkone	9	8	4	7	5	8	5	4	12	9	4	5	5	4	5	5	8	9	9	5	8	9	4	4	9
COMPARE POINTS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
UNWIND GRIPPER	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PICK DOUBLE SNAP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
appros2/check2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
movestwo	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	Y	Y	Y	Y	Y	Y	Y	Y	B	Y
checktwo	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y		Y
checktwo	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y		Y
PICK UP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y		Y
appros2/check2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y		Y
move2/check2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y		Y
checktwo	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y		Y
safe height	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y		Y
movesone	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y		Y
checksix	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y		Y

	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
NORMAL PICK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND LOCATION	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND CUBE HEIGHT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND HEIGHT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
safe height	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
move2/check2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
movesone	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
checkone	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	E
move																									
checkone																									
checkone																									
FIND CENTRE	Y	Y	Y	Y	Y		Y	Y			Y			Y			Y		Y		Y	Y			
HUNT CENTRE	2	2	2	2	1		4	2			2			2			2		2		1	2			
CHECKPRESSED	2	2	2	2	1		4	2			2			2			2		2		1	2			
FIND HEIGHT	2	2	2	2	1		4	2			2			2			2		2		1	2			
safe height	2	2	2	2	1		4	2			2			2			2		2		1	2			
move2/check2	2	2	2	2	1		4	2			2			2			2		2		1	2			
movesone	2	2	2	2	1		4	2			2			2			2		2		1	2			
checkone	2	2	2	2	1		C	2			2			2			2		2		1	2			
move	1	1	1	1			2	1			1			1			1		1			1			
checkone	1	1	1	1			2	1			1			1			1		1			1			
checkone	1	1		1			2	1			1								1			1			
COMPARE HEIGHT	1	1	2	1	1		2	1			1			2			2		1		1	2			
LOCATION	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
FIND DROP	4	4	5	4	5	4		4	5	10	4	5	4	7	5	4	5	5	6	11	7	7	9	5	
appros2/check2	4	4	5	4	5	4		4	5	10	4	5	4	7	5	4	5	5	6	11	7	7	9	5	
movesone	4	4	5	4	5	4		4	5	10	4	5	4	7	5	4	5	5	6	11	7	7	9	5	
checkone	4	4	5	4	5	4		4	5	10	4	5	4	7	5	4	5	5	D	11	7	D	9	5	
movesone	4	4	5	4	5	4		4	5	10	4	5	4	7	5	4	5	5	5	11	7	6	9	5	
checkone	4	4	5	4	5	4		4	5	10	4	5	4	7	5	4	5	5	5	11	7	6	9	5	
COMPARE POINTS	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y		Y	Y	
UNWIND GRIPPER	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y		Y	Y	
PICK DOUBLE SNAP	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y		Y	Y	
appros2/check2	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y		Y	Y	
movestwo	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y		Y	Y	
checktwo	Y	Y	Y	Y	Y	B		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y		Y	Y	
checktwo	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y		Y	Y	
PICK UP	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y		Y	Y	
appros2/check2	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y		Y	Y	
move2/check2	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y		Y	Y	
checktwo	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y		Y	Y	
safe height	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y		Y	Y	
movesone	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y		Y	Y	
checksix	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y		Y	Y	

	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
NORMAL PICK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND LOCATION	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND CUBE TOP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND HEIGHT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
safe height	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
move2/check2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
movesone	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
checkone	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
move																									
checkone																									
checkone																									
FIND CENTRE																							Y		
HUNT CENTRE																							1		
CHECKPRESSED																							1		
FIND HEIGHT																							1		
safe height																							1		
move2/check2																							1		
movesone																							1		
checkone																							1		
move																									
checkone																									
checkone																									
COMPARE HEIGHT																							1		
LOCATION	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND DROP	4	4	4	4	4	4	5	4	4	4	4	4	5	4	5	5	5	4	4	4	4	4	6	4	5
appros2/check2	4	4	4	4	4	4	5	4	4	4	4	4	5	4	5	5	5	4	4	4	4	4	6	4	5
movesone	4	4	4	4	4	4	5	4	4	4	4	4	5	4	5	5	5	4	4	4	4	4	6	4	5
checkone	4	4	4	4	4	4	5	4	4	4	4	4	5	4	5	5	5	4	4	4	4	4	6	4	5
movesone	4	4	4	4	4	4	5	4	4	4	4	4	5	4	5	5	5	4	4	4	4	4	6	4	5
checkone	4	4	4	4	4	4	5	4	4	4	4	4	5	4	5	5	5	4	4	4	4	4	6	4	5
COMPARE POINTS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
UNWIND GRIPPER	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PICK DOUBLE SNAP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
appros2/check2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
movestwo	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
checktwo	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
checktwo	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PICK UP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
appros2/check2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
move2/check2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
checktwo	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
safe height	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
movesone	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
checksix	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
NORMAL PICK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND LOCATION	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND CUBE TOP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND HEIGHT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
safe height	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
move2/check2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
movesone	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
checkone	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
move																									
checkone																									
checkone																									
FIND CENTRE					Y		Y		Y	Y			Y	Y			Y		Y		Y		Y		
HUNT CENTRE					2		2		2	3			2	3			2		2		2		2		
CHECKPRESSED					2		2		2	3			2	3			2		2		2		2		
FIND HEIGHT					2		2		2	3			2	3			2		2		2		2		
safe height					2		2		2	3			2	3			2		2		2		2		
move2/check2					2		2		2	3			2	3			2		2		2		2		
movesone					2		2		2	3			2	3			2		2		2		2		
checkone					2		2		2	3			2	3			2		2		2		2		
move					1		1		1	1			1	1			1		1		1		1		
checkone					1		1		1	1			1	1			1		1		1		1		
checkone					1		1		1	1			1	1			1		1		1		1		
COMPARE HEIGHT					1		1		1	1			1	2			1		1		1		1		
LOCATION	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FIND DROP	7	6	7	6	6	7	6	7	1	7	6	6	7	7	6	6	6	6	6	6	6	6	1	7	6
appros2/check2	7	6	7	6	6	7	6	7	1	7	6	6	7	7	6	6	6	6	6	6	6	6	1	7	6
movesone	7	6	7	6	6	7	6	7	1	7	6	6	7	7	6	6	6	6	6	6	6	6	1	7	6
checkone	7	6	7	6	6	7	6	7	A	7	6	6	7	7	6	6	6	6	6	6	6	6	A	7	6
movesone	7	6	7	6	6	7	6	7		7	6	6	7	7	6	6	6	6	6	6	6	6		7	6
checkone	7	6	7	6	6	7	6	7		7	6	6	7	7	6	6	6	6	6	6	6	6		7	6
COMPARE POINTS	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
UNWIND GRIPPER	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
PICK DOUBLE SNAP	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
appros2/check2	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
movestwo	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
checktwo	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
checktwo	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
PICK UP	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
appros2/check2	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
move2/check2	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
checktwo	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
safe height	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
movesone	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
checksix	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y

Appendix E

Pick Run Tables

The following tables detail the runs of the experimental robotic assembly system described in Chapter 6. These runs involve the part acquisition Behavioural Modules detailed in Section 6.1 of the above chapter. The hierarchical structure for the Behavioural Modules can be found in Chapter 6, Fig 6–4.

The run numbers head the columns of the table, and the row headers show the Behavioural Module names which are used. The ones picked out in capital letters are the Behavioural Modules in upper level of the hierarchy, with the others forming the lowest level Behavioural Modules in the structure. The Behavioural Modules not on the *Ideal Execution Path* are indented. The numbers shown against the **Hunt Centre** Behavioural Module and those below show the number of probes it took to find the top of the cube, having already found the chamfer. The numbers shown against the **Find Drop** Behavioural Module refer to the number of horizontal sweeps across the cube which have been made to determine the orientation of the cube. The actual errors which occurred are noted by the addition of a letter corresponding to the explanation of these given in Section 6.1.

Appendix D

Soma5a

This diagram also uses the spine representational form described in Appendix C to show the sub-set of the Soma-5 set which is referred to as the Soma-5a group.

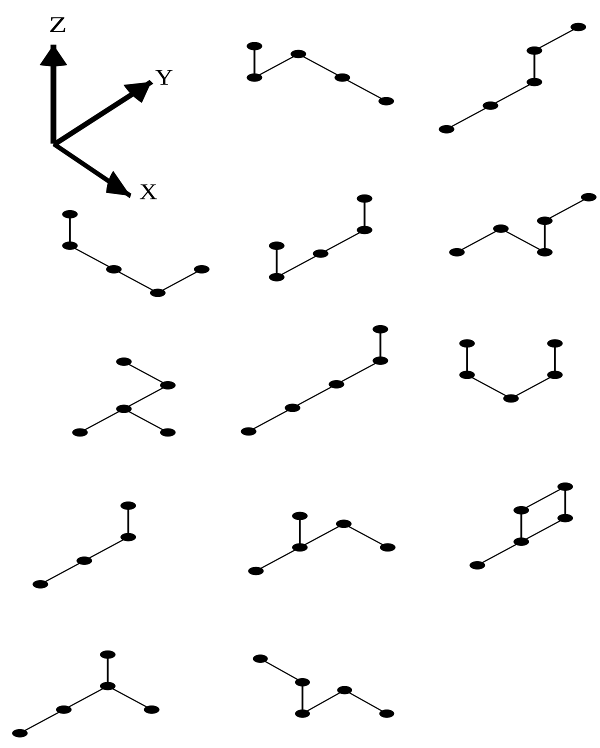


Figure D–1: Spine Representation the Soma-5a group.

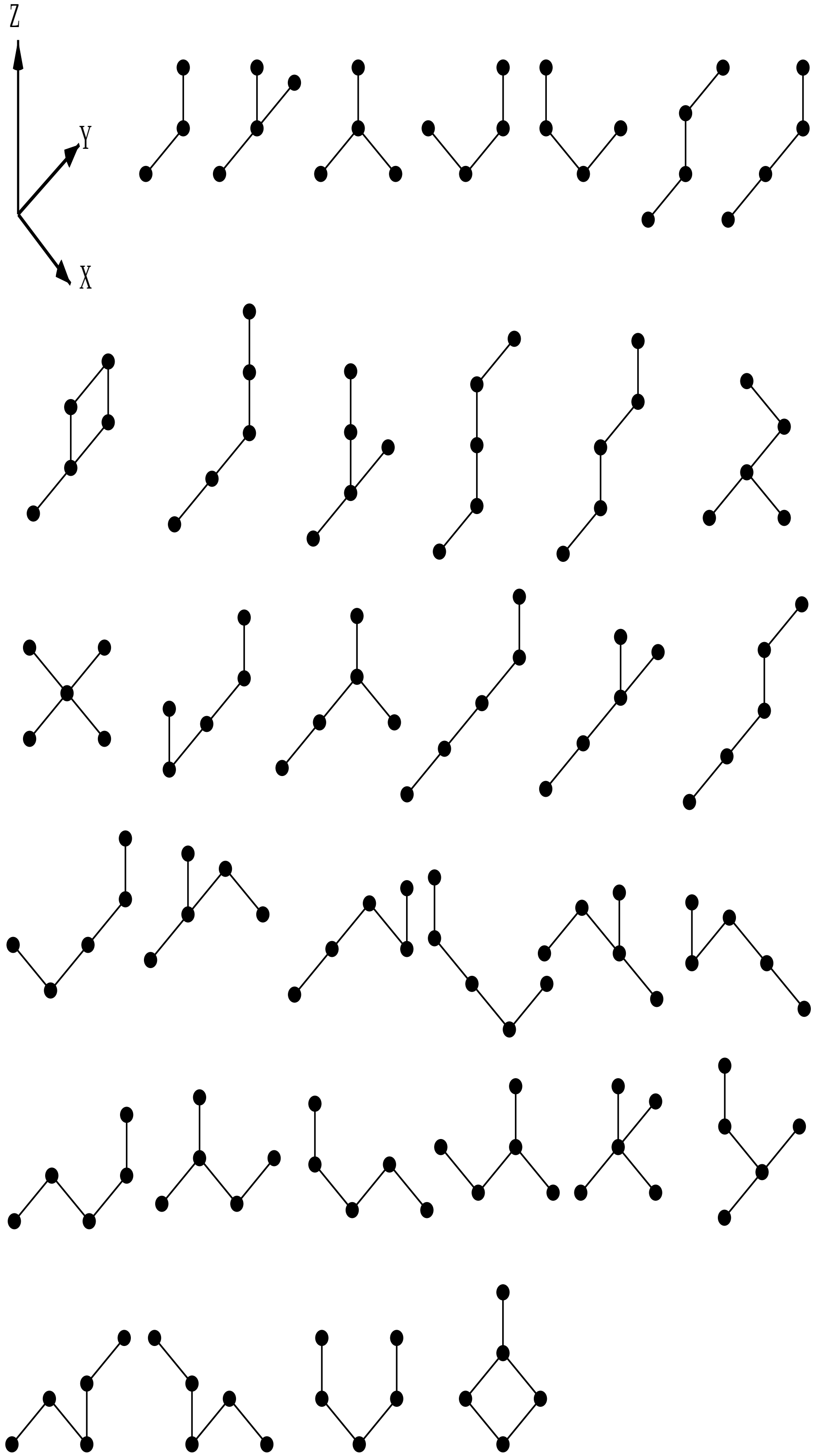


Figure C–1: A Spine Representation of the Soma-5 set.

Appendix C

Spine Representation of the Soma-5 Set

These diagrams have been produced using the representation of shrinking a Soma cube down to a single point, and connecting these points by lines to form the Soma shapes. This fits in well with the idea of representing Soma shapes in Prolog by a list of [x,y,z] triples.

The Soma-5 set comprises of the one Soma-3 shape, the six Soma-4 shapes and twenty-eight Soma-5 shapes. Each of these are shown on the following page in spine representation form.

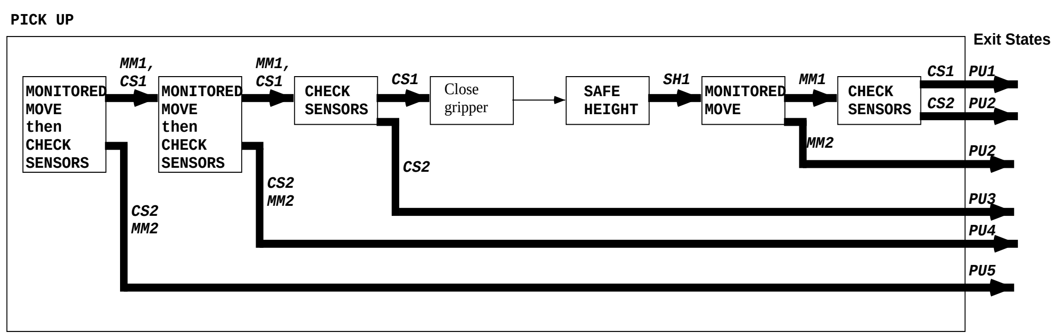


Figure B–18: Pick Up Behavioural Module

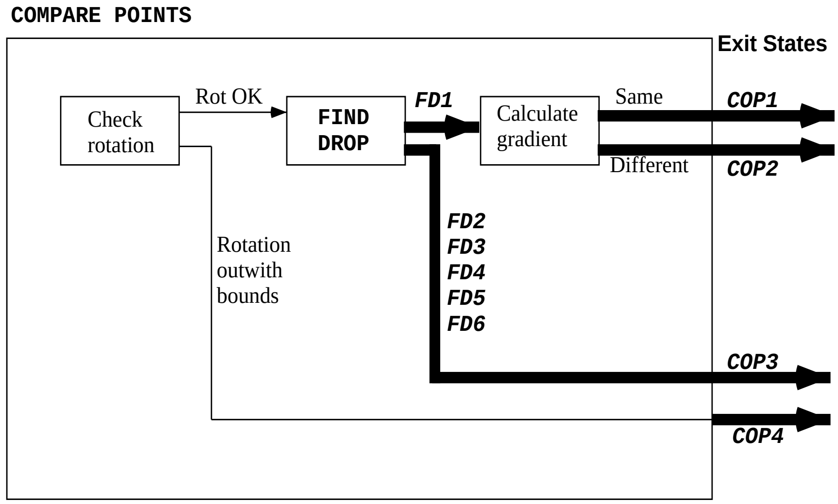


Figure B–15: Compare Points Behavioural Module

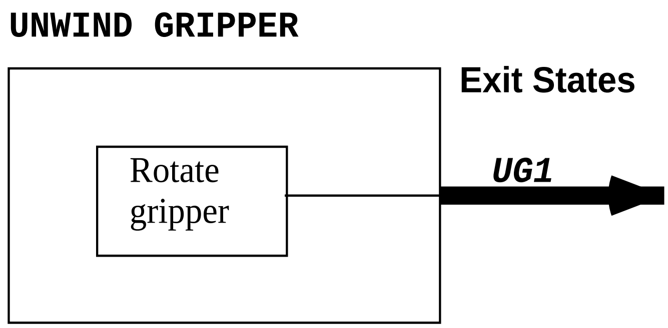


Figure B–16: Unwind Gripper Behavioural Module

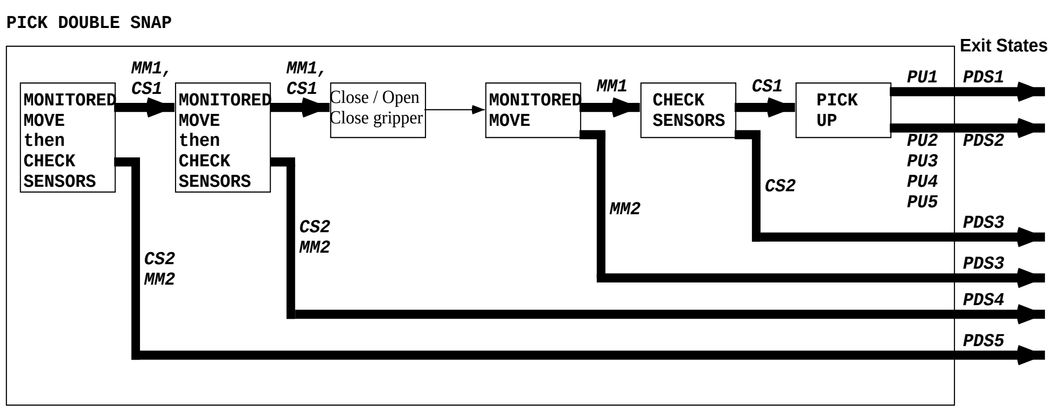


Figure B–17: Pick Double Snap Behavioural Module

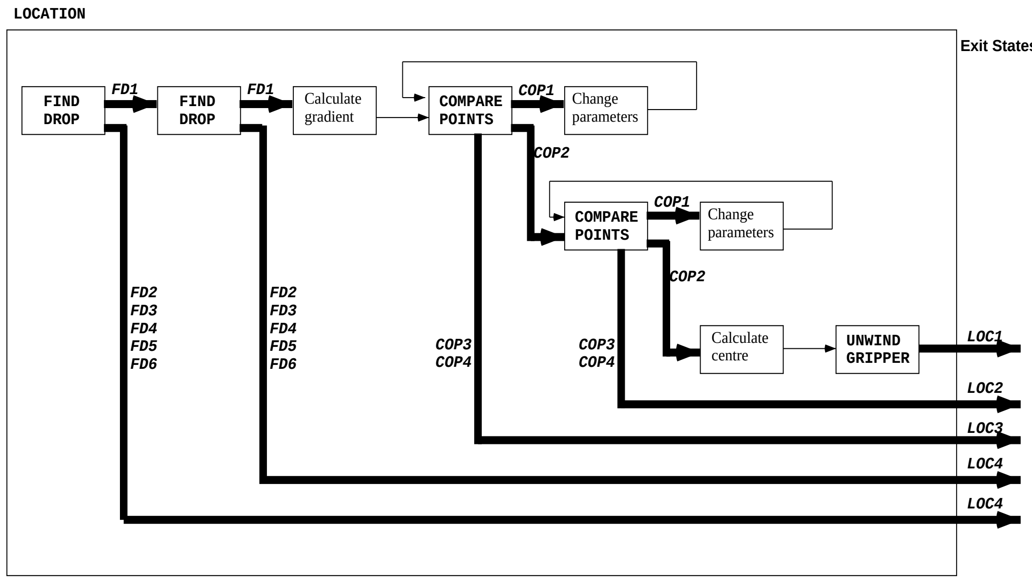


Figure B–13: Location Behavioural Module

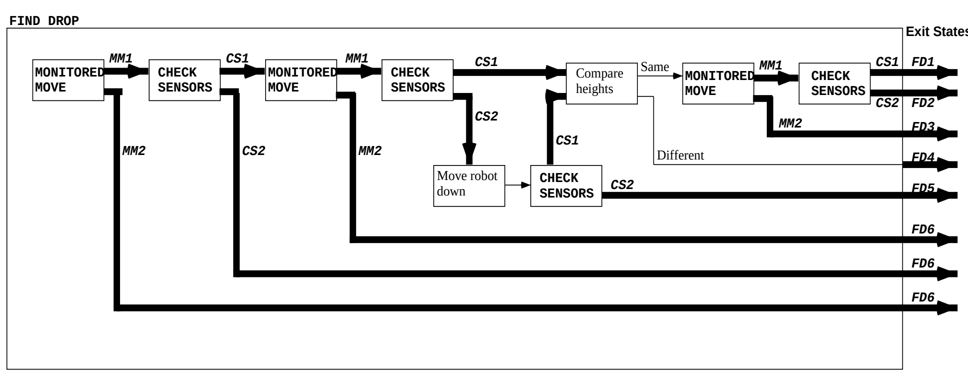


Figure B–14: Find Drop Behavioural Module

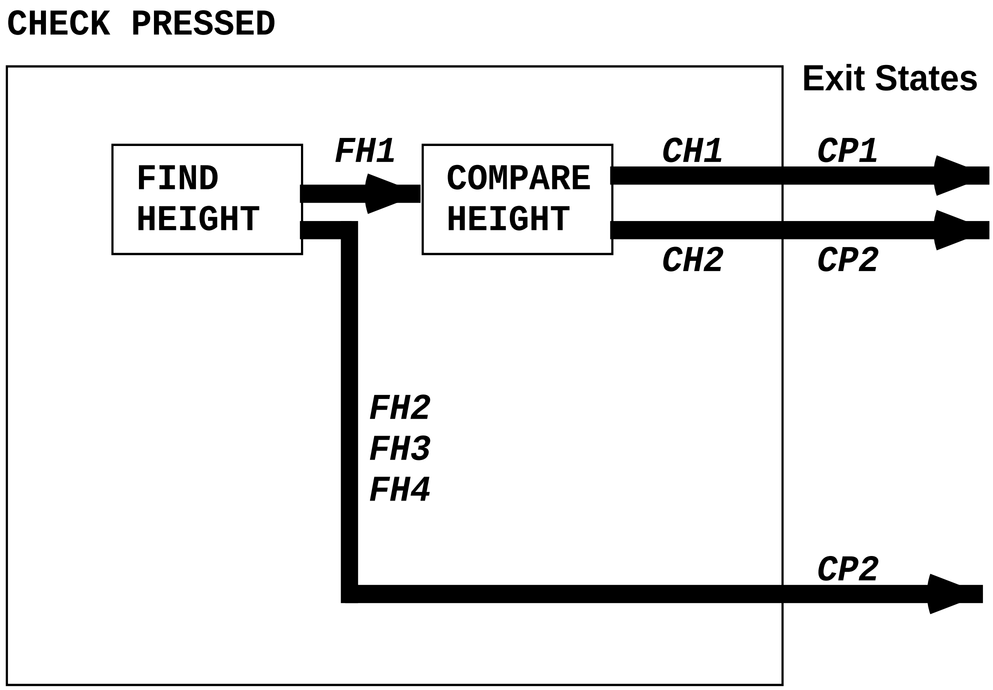


Figure B–11: Check Pressed Behavioural Module

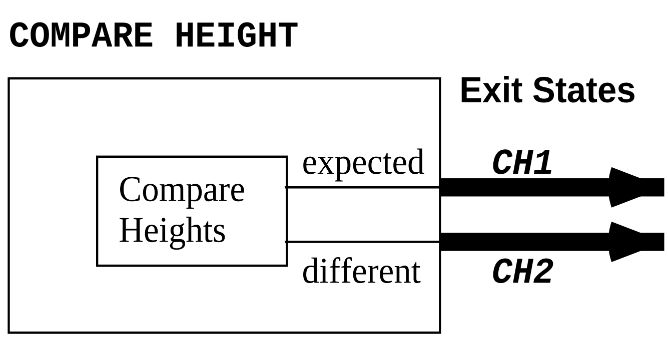


Figure B–12: Compare Height Behavioural Module

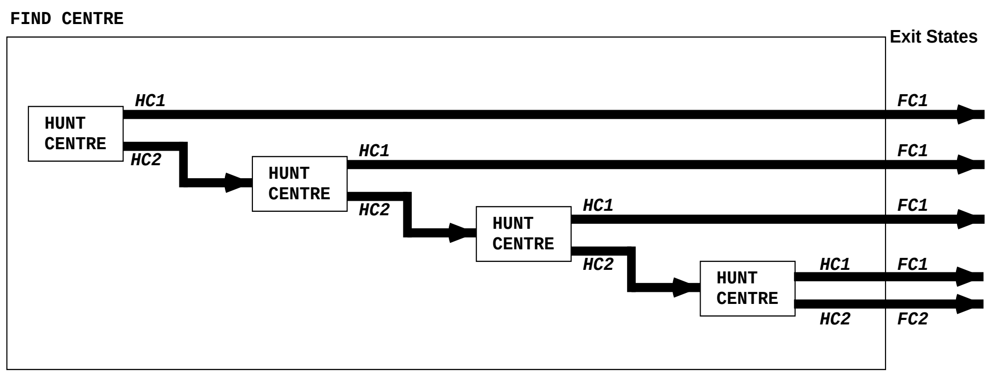


Figure B–9: Find Centre Behavioural Module

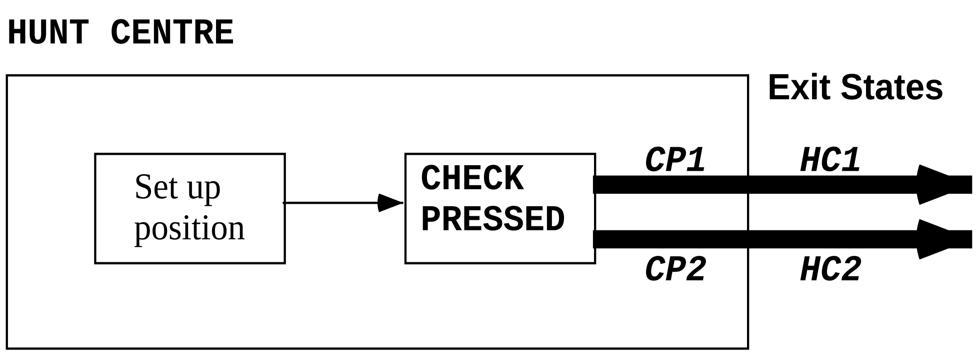


Figure B–10: Hunt Centre Behavioural Module

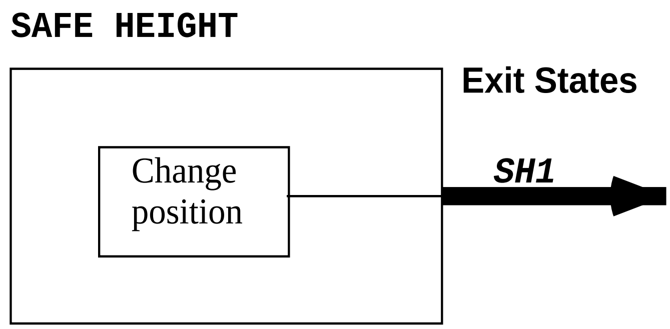


Figure B–6: Safe Height Behavioural Module

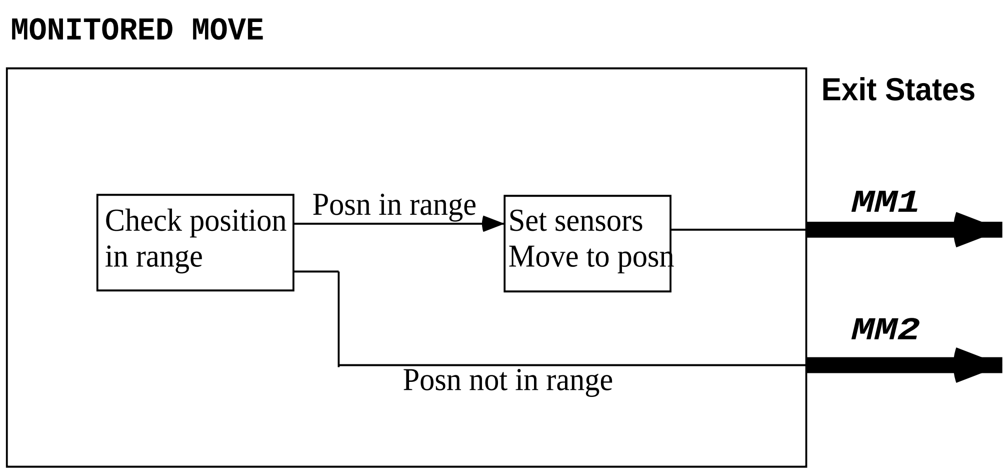


Figure B–7: Monitored Move Behavioural Module

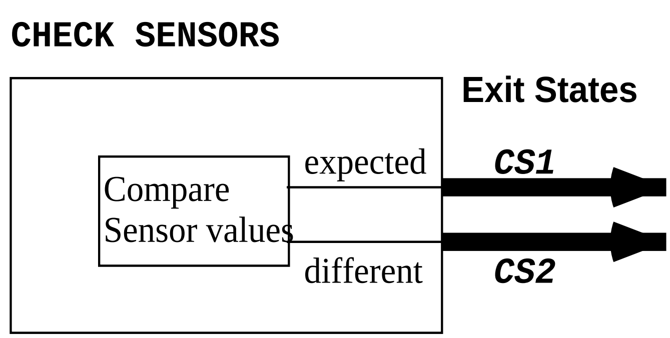


Figure B–8: Check Sensors Behavioural Module

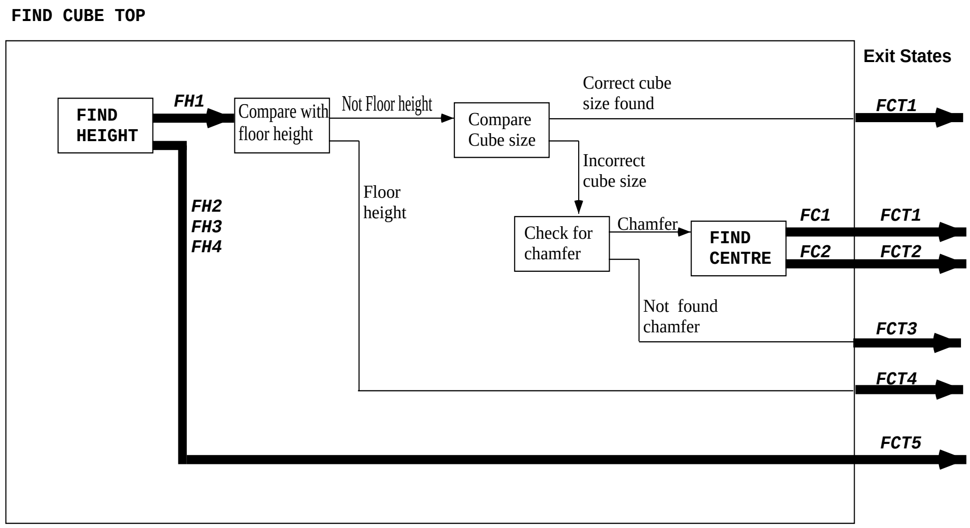


Figure B–4: Find Cube Top Behavioural Module

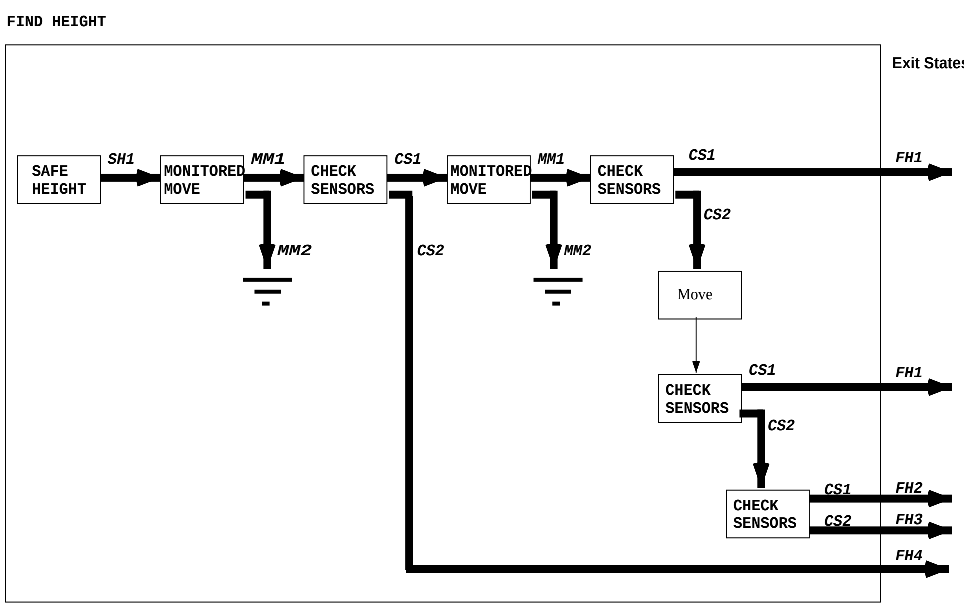


Figure B–5: Find Height Behavioural Module

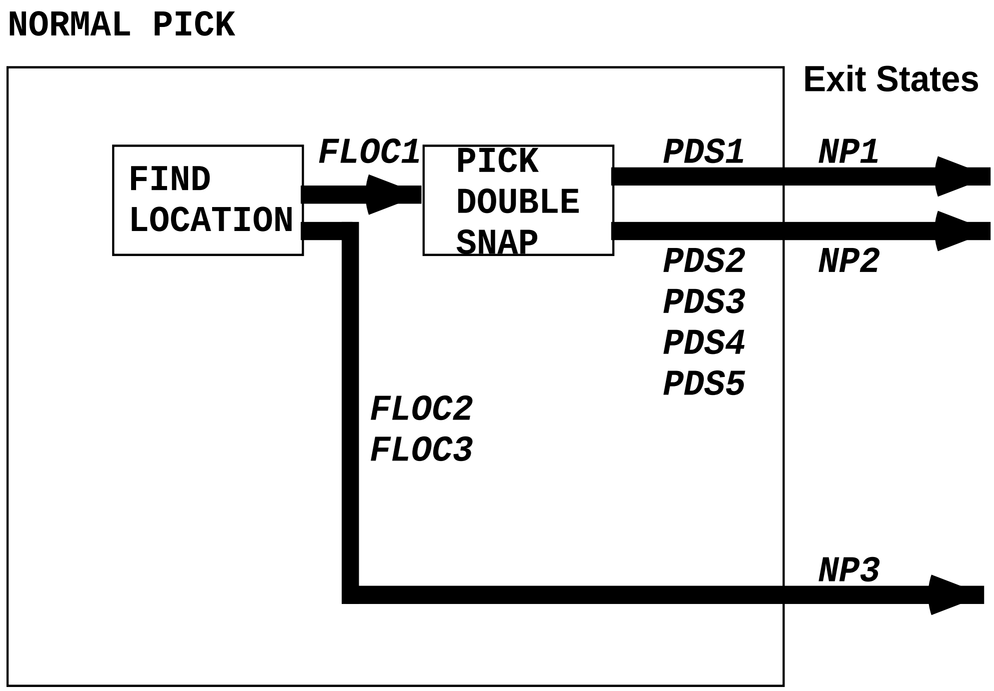


Figure B–2: Normal Pick Behavioural Module

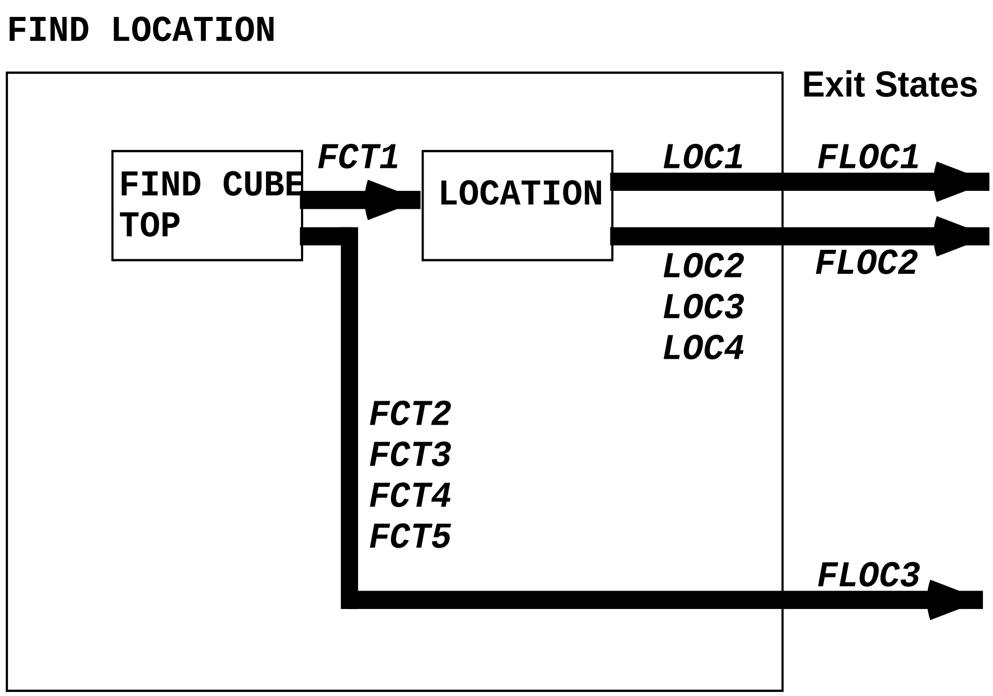


Figure B–3: Find Location Behavioural Module

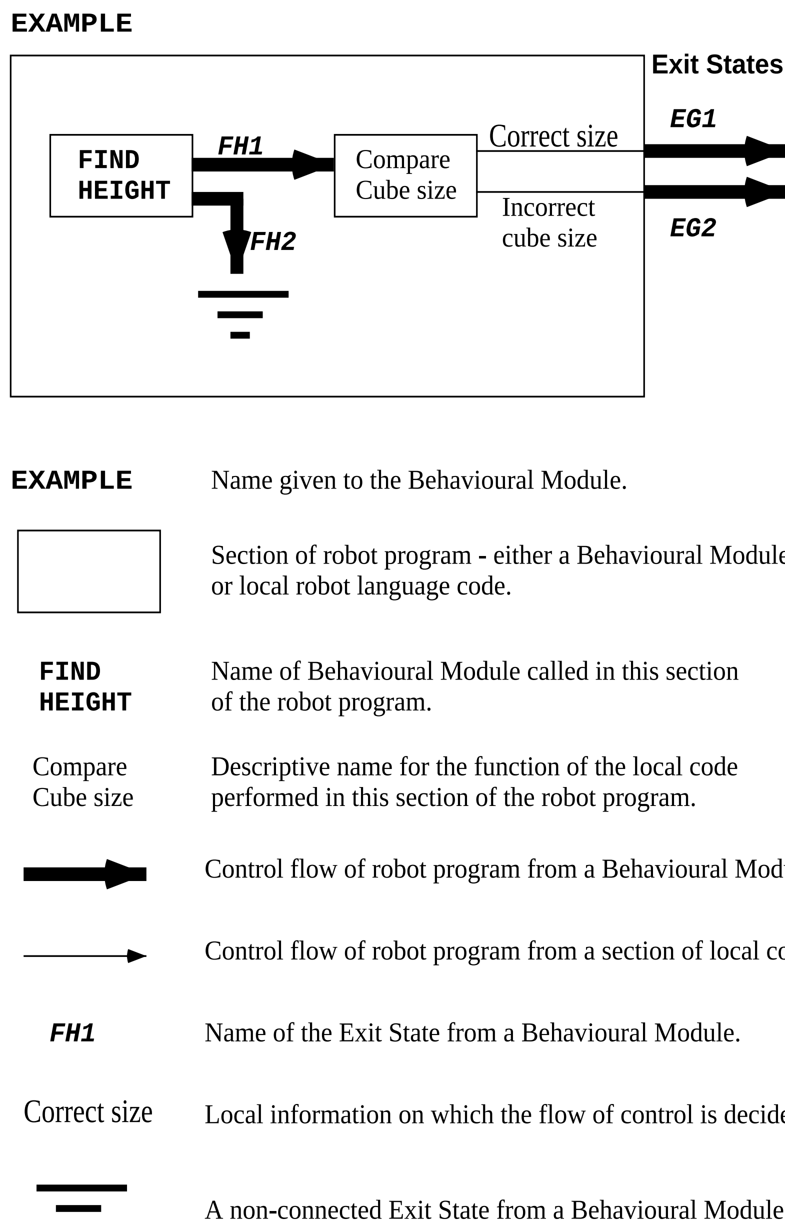


Figure B–1: Key for Behavioural Module Diagrams.

Monitored Move (Fig B–7) Monitors the robot gripper sensors while the robot is moving, halting the robot if any of the sensors are triggered.

Check Sensors (Fig B–8) Checks the actual values of the binary sensors against given values.

Find Centre (Fig B–9) Calls Behavioural Modules to perform a simple search strategy to find the top of a cube, given that a chamfer has already been found.

Hunt Centre (Fig B–10) Simple search strategy which involves testing heights found in each of four directions.

Check Pressed (Fig B–11) Calls Behavioural Modules to find a height and check if it corresponds to the top of the cube.

Compare Height (Fig B–12) Checks if a height has been found corresponding to the top of a cube.

Location (Fig B–13) Calls Behavioural Modules to determine the orientation of the part.

Find Drop (Fig B–14) Moves horizontally over the top of the cube till the edge is detected.

Compare Points (Fig B–15) Calls Behavioural Modules which check the values produced by the orientation strategy.

Unwind Gripper (Fig B–16) Rotate gripper to undo wrist wind-up.

Pick Double Snap (Fig B–17) Calls Behavioural Modules to pick up part using a double snap of the gripper.

Pick Up (Fig B–18) Picks up the part using a single snap.

Appendix B

Behavioural Module Diagrams

This Appendix provides Behavioural Module Diagrams which are used for the experimental system. The key for the diagrams has been provided again (see Fig B–1) for convenience, although it also appears in Chapter 4.

The following descriptions detail briefly the function of each Behavioural Module shown in the following diagrams:

Normal Pick (Fig B–2) Calls Behavioural Modules to locate the Soma part and then pick it up.

Find Location (Fig B–3) Calls Behavioural Modules to find top of the cube to be gripped and then calls Behavioural Modules to determine the orientation of the cube.

Find Cube Top (Fig B–4) Calls Behavioural Modules to find a point on the top of the cube. If a chamfer height is found instead, calls a Behavioural Module to attempt to locate the actual top of the cube.

Find Height (Fig B–5) Calls low level Behavioural Modules to return a height found using the binary probe sensors.

Safe Height (Fig B–6) Finds a safe height in the workcell which is within reach of the robot.

```
[left,                                      ; part name
[[[0,0,0],[0,1,0],[1,1,0],[1,1,1]],         ; nominal position
[[0,x,z],[1,1,1]]],                         ; alignment & gripped cube (pick)
[[[z,1]],                                   ; gripper/wrist alignment(regrasp)
[[0,0,0],[0,1,0],[0,1,1],[1,0,0]],          ; nominal posn on regrasp table
[[0,0,0],[0,1,0]],                          ; base offset on regrasp table
[[0,y,z],[0,1,1]],                          ; alignment and gripped cube (put)
[[0,y,-(x)],[0,1,1]],                       ; alignment and gripped cube (get)
[[y,1],[z,0]]],                             ; rotation for place into assembly
[1,1],                                      ; padding offsets
[[[z,1],[y,1]],[[1,1,1],                    ; rotation and translation (place)
[[[1,1,1],[1,1,2],[1,2,2],[2,2,2]],         ; nominal position in assembly
[[0,y,z],[2,2,2]]]]]],                      ; alignment & gripped cube (place)

[fork3,                                     ; part name
[[[0,1,0],[1,0,0],[1,1,0],[1,1,1]],         ; nominal position
[[0,x,z],[1,1,1]]],                         ; alignment & gripped cube (pick)
[[[z,3],[y,3]],                             ; gripper/wrist alignment(regrasp)
[[0,0,1],[1,0,0],[1,0,1],[1,1,1]],          ; nominal posn on regrasp table
[[1,0,0]],                                  ; base offset on regrasp table
[[0,y,-(x)],[0,0,1]],                       ; alignment and gripped cube (put)
[[0,y,z],[0,0,1]],                          ; alignment and gripped cube (get)
[[z,0]]],                                   ; rotation for place into assembly
[2,0],                                      ; padding offsets
[[[z,3],[y,3]],[[1,0,1],                    ; rotation and translation (place)
[[[1,0,2],[2,0,1],[2,0,2],[2,1,2]],         ; nominal position in assembly
[[-,y,z],[1,0,2]]]]]]]                      ; alignment & gripped cube (place)
```

```
[[0,y,z],[1,0,1]]],                     ; alignment & gripped cube (pick)
[],                                     ; regrasp details
[2,1],                                  ; padding offsets
[[[z,3]],[[2,0,0],                      ; rotation and translation (place)
[[[2,0,0],[2,1,0],[2,1,1],[2,2,1]],     ; nominal position in assembly
[[0,x,z],[2,1,1]]]]]],                  ; alignment & gripped cube (place)

[right,                                 ; part name
[[[0,0,0],[0,1,0],[1,0,0],[1,0,1]],     ; nominal position
[[0,y,z],[1,0,1]]],                     ; alignment & gripped cube (pick)
[[[z,0],[y,3]],                         ; gripper/wrist alignment(regrasp)
[[0,0,1],[1,0,0],[1,0,1],[1,1,0]],      ; nominal posn on regrasp table
[[1,0,0]],                              ; base offset on regrasp table
[[0,y,-(x)],[0,0,1]],                   ; alignment and gripped cube (put)
[[0,y,z],[1,0,1]],                      ; alignment and gripped cube (get)
[[z,0]]],                               ; rotation for place into assembly
[1,0],                                  ; padding offsets
[[[z,0],[y,3]],[[0,0,0],                ; rotation and translation (place)
[[[0,0,1],[1,0,0],[1,0,1],[1,1,0]],     ; nominal position in assembly
[[0,y,z],[1,0,1]]]]]],                  ; alignment & gripped cube (place)

[ell,                                   ; part name
[[[0,0,0],[0,0,1],[1,0,0],[2,0,0]],     ; nominal position
[[0,y,z],[0,0,1]]],                     ; alignment & gripped cube (pick)
[[[z,1],[x,3]],                         ; gripper/wrist alignment(regrasp)
[[0,0,0],[0,0,1],[0,0,2],[0,1,2]],      ; nominal posn on regrasp table
[[0,0,0]],                              ; base offset on regrasp table
[[0,x,y],[0,1,2]],                      ; alignment and gripped cube (put)
[[0,x,-(y)],[0,0,1]],                   ; alignment and gripped cube (get)
[[x,3],[z,0]]],                         ; rotation for place into assembly
[0,2],                                  ; padding offsets
[[[z,1],[x,2]],[[0,0,1],                ; rotation and translation (place)
[[[0,0,2],[0,1,2],[0,2,1],[0,2,2]],     ; nominal position in assembly
[[0,x,z],[0,1,2]]]]]],                  ; alignment & gripped cube (place)
```

Appendix A

Prolog List for Soma Assembly

The following Prolog list represents a complete run of the assembly with seven pick and place actions. A full explanation of one of these pick/place sub-lists is given in Chapter 3 on page 69.

```
[[lell,                                   ; part name
[[[0,0,0],[0,0,1],[1,0,0]],               ; nominal position
[[0,y,z],[0,0,1]]],                       ; alignment & gripped cube (pick)
[],                                       ; regrasp details
[0,1],                                    ; padding offsets
[[[z,3]],[[0,0,0],                        ; rotation and translation (place)
[[[0,0,0],[0,1,0],[0,1,1]],               ; nominal position in assembly
[[0,x,z],[0,1,1]]]]]]],                   ; alignment & gripped cube (place)

[fork2,                                   ; part name
[[[0,0,0],[1,0,0],[1,0,1],[2,0,0]],       ; nominal position
[[0,x,z],[1,0,1]]],                       ; alignment & gripped cube (pick)
[],                                       ; regrasp details
[1,2],                                    ; padding offsets
[[[z,2]],[[0,2,0],                        ; rotation and translation (place)
[[[0,2,0],[1,2,0],[1,2,1],[2,2,0]],       ; nominal position in assembly
[[0,x,z],[1,2,1]]]]]]],                   ; alignment & gripped cube (place)

[zed,                                     ; part name
[[[0,0,1],[1,0,0],[1,0,1],[2,0,0]],       ; nominal position
```

In the longer term, learning could be introduced into the type of architecture detailed in this thesis. At present, all the design decisions are based on human knowledge, but this is an area in which the system itself could be used to learn where an Alternative Execution Path may be useful, and where the Behavioural Modules should be inserted.

From the above experimental changes, an investigation could be made into what constitutes a good Behavioural Module, and what type of competences are required. A useful expansion of this type of system would be to find out if these low level Behavioural Modules provide a useful set of competences which can also be used to build more complex systems. As yet, not enough Behavioural Modules have been designed to examine the question of what makes a good Behavioural Module and good modular boundaries, but it is hoped that the above system can provide a basic framework for investigating this matter.

7.3.2 Long Term

As has been mentioned above, so far in the development of these Behaviour-based systems, not many Behavioural Modules have been designed and implemented. An interesting line of research would be to determine what low level competences are necessary from which generalised systems could be developed. This type of work could lead to the eventual goal of providing a library of Behavioural Modules from which the programmer could select useful ones for a particular application.

This leads to the development of support tools to develop the Behavioural Modules and link them together. As Behavioural Modules would thus be re-used in different situations, a more formalised description of their competence would need to be developed, and part of this could be formed from the known Exit States and the control paths through the Behavioural Modules. If a library of already tested and reliable Behavioural Modules were available, this would reduce the time currently needed to develop an assembly system. Even a library of Behavioural Modules with a low level of competence would be useful as it has been shown that these can be easily linked to provide a system of greater competence. A useful decoupling of the development of Behavioural Modules from their integration into a system can thus be defined, allowing the Behavioural Modules to be updated independently while system development can be continuing.

7.3 Further Work

In this section, some outlines for further research are suggested. These are split into two sections: the first examines improvements to the current execution system and the second looks at longer term work.

7.3.1 System Extensions

So far, the above ideas have only been tested in a relatively simple system with binary sensors and limited complexity. Many useful Behavioural Modules have been developed, but it has yet to be shown whether this methodology can generalise out into more complex systems tackling more complicated assemblies.

Within the experimental domain which has been developed, new strategies could be implemented to improve the reliability of the system. This could be done in the first instance by enhancing individual Behavioural Modules, and could also include better alternative strategies should some of the Behavioural Modules on the Ideal Execution Path go wrong. The use of the Soma world could also be expanded to include different strategies to deal with some of the assemblies which are currently impossible, such as assemblies with cyclic adjacency and overhang problems.

Along with developing Behavioural modules with different strategies to expand the set of Behavioural Modules, an investigation could usefully be made into the use of different and more complex sensors. The current Behavioural Modules only use simple binary input from sensors, and more complex problem, such as how to deal with more complicated monitoring of the Behavioural Modules should be investigated in the context of Exit States, the Ideal Execution Path, and Alternative Execution Paths. Different industrial assemblies could be attempted, and different robots used to examine the generality of this methodology.

ate, allowing easy insertion of Alternative Execution Paths. The Behavioural Modules can be designed to limit the complexity of the individual Behavioural Modules, without limiting the complexity of the system.

Retaining the context of the task

Unlike in systems such as the Error Detection and Recovery schemes, the architecture discussed in this thesis retains the context in which the potential failures occur. As the Behavioural Modules are designed, built and tested in the real assembly workcell, the software and hardware can be specifically matched with the task in hand. This allows the designer of the system to correctly identify the problems and provide appropriate alternative if necessary. The context for the Behavioural Module task will be present as the system is run, rather than having the information extracted from the system to be dealt with off-line.

Behavioural Module Examination

The system can provide a framework for looking at what constitutes a good Behavioural Module. The above system architecture can be used to examine how well a particular Behavioural Module is monitored by sensory coverage and to detect all possible control paths within it to see if it has successfully completed its task. The system can be run in the real world and the output from the system can be closely examined. Once a system has been developed and appropriate Behavioural Modules inserted, the system can still be examined while running to see if, for example, Alternative Execution Paths have been used more often than normal, and therefore something may be misaligned in the actual workcell. The output from this type of system can also be examined to check the suitability of the Behavioural Modules on the Ideal Execution Path to perform the assembly.

for a task, instead of being guaranteed to correctly perform it. This means that, unlike in many other systems where the system architecture can actually degrade the reliability, this type of system uses the architecture to increase it.

Complete on-line reliability

Due to the way in which the system has been designed, it is able to contain the problem of variation and uncertainty in the on-line execution system. The planner is freed from having to reason about variation and uncertainty, and therefore all this type of physical workcell detail does not have to be transported off-line.

The system itself has no notion of specifically detecting errors, diagnosing the problem, and performing recovery actions. Instead, it just keeps trying to perform the task, sometimes modifying its actions according to the actual circumstances. It is left to the designer of the system to insert appropriate alternative actions if deemed necessary.

The system design allows these alternatives to be inserted at all levels of the Behavioural Module hierarchy, with no outright failure until the top of the hierarchy is reached. This allows the system great flexibility to deal with the problem depending on what sensors and strategies it has available. An attempt can be made to rectify a particular failure at a low level, but if this does not succeed, it may still be possible to use an alternative at a higher level.

As the planner provides only part motions, and does not dictate how these will be performed (unlike some systems where each robot motion is sent down from the planner), the execution system has complete freedom to interpret these part motions according to what Behavioural Modules are available. This type of system can provide the flexibility to change old strategies with new and improved ones without changing the interface to the planner. This allows full use to be made of introducing new sensors into the system, or designing new strategies to perform a sub-task. The hierarchical structure of this type of system also allows Behavioural Modules to be slotted in where appropri-

States. New strategies were incorporated using the sensors which have been detailed in previous chapters. The execution system was built up from low level, low competence, Behavioural Modules, and, as each level of the hierarchy was formed, the Behavioural Modules were tested to check that the hardware and the software were well integrated to perform the given task. As the system developed in this way, it was a relatively easy task to identify possible error situations, and, for some of these cases, it was possible to slot in an Alternative Execution Path to deal with the problem should it arise in the actual assembly. The building of Alternative Execution Paths could be done at any level of the hierarchy wherever useful alternative Behavioural Modules were available. The places where possible error situations could occur were identified due to the close relation between sensing, action, software, and hardware, within the experimental Behavioural Modules. These can also be recognised quickly, allowing the system to be immediately halted or perform alternative actions, rather than going on to perform inappropriate motions.

7.2.1 The Thesis Contribution

In this thesis, a system has been developed and implemented which is designed to increase the reliability of a Behaviour-based robotic assembly system. In doing so, several other points of significance have been discovered. This section outlines the contributions of this research.

Relaxed Reliability of Individual Behavioural Modules

The kind of modular structure which has been developed for the thesis has also shown that the constraints formerly imposed on Behaviour Modules, requiring them to be very reliable task-achievers, can now be relaxed. Although, as far as the system design is concerned, individual modular reliability is still a useful thing to aim for, it can be seen that the system is forgiving of individual failures and thus allows more flexibility in how reliable the individual Behavioural Modules must be. Therefore, a Behavioural Module can now be good enough

To enable these choices to be made, the following terms have been introduced: Exit States, the Ideal Execution Path, and Alternative Execution Paths.

The concept of one returned parameter (possibly qualified by other parameters), called an Exit State, which shows the internal control path of a Behavioural Module, proved to be useful when determining the linking of Behavioural Modules. The use of other local information which provided control decision points within Behavioural Modules were also used as determining factors as to whether a Behavioural Module had performed as required or whether some alternative action should be performed to keep the system from failing. The Exit State value for a particular Behavioural Module is determined from local data, sensor data, and from Exit States returned from lower level Behavioural Modules.

Two other useful concepts have been introduced in the thesis work: the Ideal Execution Path and Alternative Execution Paths. These define control paths through the hierarchy of the Behavioural Modules. The paths are determined from the Exit States of Behavioural Modules and from decision points in the local robot code used within the Behavioural Modules. The Ideal Execution Path describes the best route, according to the system designer, through the Behavioural Modules performing the assembly. Even though the Behavioural Modules are designed and tested to be reliable, it can be seen that they are not always perfect at performing the assembly task, and therefore, alternative Behavioural Modules, those on an Alternative Execution Path can be included to restore the system to the Ideal Execution Path. Alternative Execution Paths can be brought into use when the Behavioural Modules on the Ideal Execution Path cannot deal with the actual variation and uncertainty in the workcell, or when one has failed while performing its task.

An experimental system has been designed and implemented to test the above ideas, and examine the internal and external linking of the system. Simple sensors were integrated into the system to monitor the Behavioural Modules and to be used to help determine more accurate information about the real world in which the assembly is performed, and thus provide valid Exit

and are dealt with manually. The first is by far the more popular method, but this is only feasible when dealing with products which have the advantage of volume. Research into the problem has been divided into three categories: Error Detection and Recovery systems which rely on off-line knowledge sources to re-plan on failure, Worst Case Analysis systems which concentrate on producing plans which are guaranteed to succeed, and Competent Execution Systems which rely on the run-time component to deal with variation and uncertainty.

7.2 Summary of the thesis work

The thesis work looks at increasing the reliability of an example of the third type of system mentioned above, the Competent Execution System. The Behaviour-based SOMASS system's reliability comes from implemented variation reducing motion strategies in single Behavioural Modules (such as sweeping the parts into the nominal position for picking them up), variation containment strategies across several Behavioural Modules (such as leaving gaps in the assembly to be removed by patting the shape together at the end of the complete assembly), and by providing information from the planner conducive to performing an action in one way (such as the grip positions given on the cubes dictating that the part should be gripped from above).

Following this Behaviour-based approach, the work of the thesis has sought to extend the scope of the reliability of the system to include a reliable system architecture. This work has shown that with the aid of sensors and local information, it can be determined whether a sub-task has been correctly performed within the context of its calling or whether it has failed to deal with the actual workcell conditions. It has been shown that if a failure is discovered, or the workcell conditions are such that the Behavioural Modules cannot actually perform the task, alternative actions may be available, determined by the system designer, which will prevent the system from falling into an error situation.

Chapter 7

Conclusions and Further Work

This thesis has been concerned with introducing sensors into a Behaviour-based robotic assembly system and investigating how these can be used to improve its overall reliability. The problem of achieving both a reliable and easily programmed assembly system has been the focus of much research in the area of robotics but, as yet, no definitive answer has been found.

The method of achieving reliability demonstrated in the Behaviour-based SOMASS system has been expanded to encompass Behavioural Modules embedded in a reliable system architecture. This new approach still retains the modularity which helps the programmer to construct new applications and the method of linking the Behavioural Modules does not hinder this ability.

In this chapter, the main features of the research will be summarised, with the main contributions indicated, then areas for further research will be listed.

7.1 Summary

As has been identified in the previous chapters, one of the major problems facing users of robotic assembly systems is providing a reliable system which is also easy to program. Current systems in use in industry tend to involve a trade off between the two: either the workcell is tightly engineered to prevent variation causing many failures, or the error situations are allowed to happen

A useful way of keeping the system running at the edge of the competence of the Behavioural Modules has also been shown. If the system uses Alternative Execution Paths more frequently than expected or previously found, the system can be examined to determine the cause of this unexpected use while still running.

I Part initially acquired in wrong orientation due to too much orientation variation when setting it in the workcell.

J Robot invalid joint position error.

K One probe remained stuck in the 'on' position.

L Wrong probe sensor triggered in initial downward probe.

M Part unstable on the re-grasp table.

Each of these problems were identified by the Exit States of the system, apart from the hardware faults where communication with the robot failed completely and the whole system had to be restarted. For each of the other error situations, possible Alternative Execution Paths could be provided if required.

6.3 Summary of assembly runs

This chapter has detailed some assembly runs which were performed using the assembly system described in Chapter 5.

The results show that, even when using Behavioural Modules which are unreliable, so long as an appropriate Exit State can be returned from these modules and other local information is available, the system can be made more reliable by the introduction of Alternative Execution Paths.

In all of the above runs, the failure of the system was detected at a low level, except in the case of hardware faults. The structure of the system also meant that alternative actions could be made available at any level of the hierarchy. For all of the above error situations, it is possible to devise strategies or new Behavioural Modules to replace those which fail if required. It may of course be the case that there is no alternative which can be devised, in which case, operator intervention is required.

(entered 4 times), and **Move Round Assembly** (entered 45 times). All these Behavioural Modules are described in Chapter 5, Section 5.1.3.

Out of the 103 runs of the system, 79 produced correct assemblies, with the 24 failures due to individual failures in the pick/place Behavioural Modules. The specific failures are shown in the digrams in Figs 6–14, 6–15, and 6–16. Of the total of 678 individual pick/place for individual parts, 654 were successfully performed (accounting for the 24 failed whole runs above). The Ideal Execution Path follows the leftmost control path vertically downwards, with the number of runs following these also shown. The actual causes of the failed runs are detailed in the following description. Where they occurred is shown both in the figures and in the detailed runs of the assembly in Appendix F.

A As for failure 'A' for the part acquisition runs. The **Find Height** Behavioural Module found a point right on the edge of the top just on the chamfer, and thus when the **Location** Behavioural Module returns to this point, occasionally the sensor pushes the cube if trying to move horizontally.

B Hardware fault.

C As for failure 'B' for the part acquisition runs. Incorrect orientation calculation — detected as the fingers move over the part to be picked up.

D Part pushed aside by trailing probe during the strategy to find the orientation of the cube.

E Part slipped in the gripper and could not be re-grasped.

F Part rotated on re-grasp table and could not be re-acquired.

G Part slipped in gripper while moving round the assembly and could not be re-acquired.

H Part dropped completely.

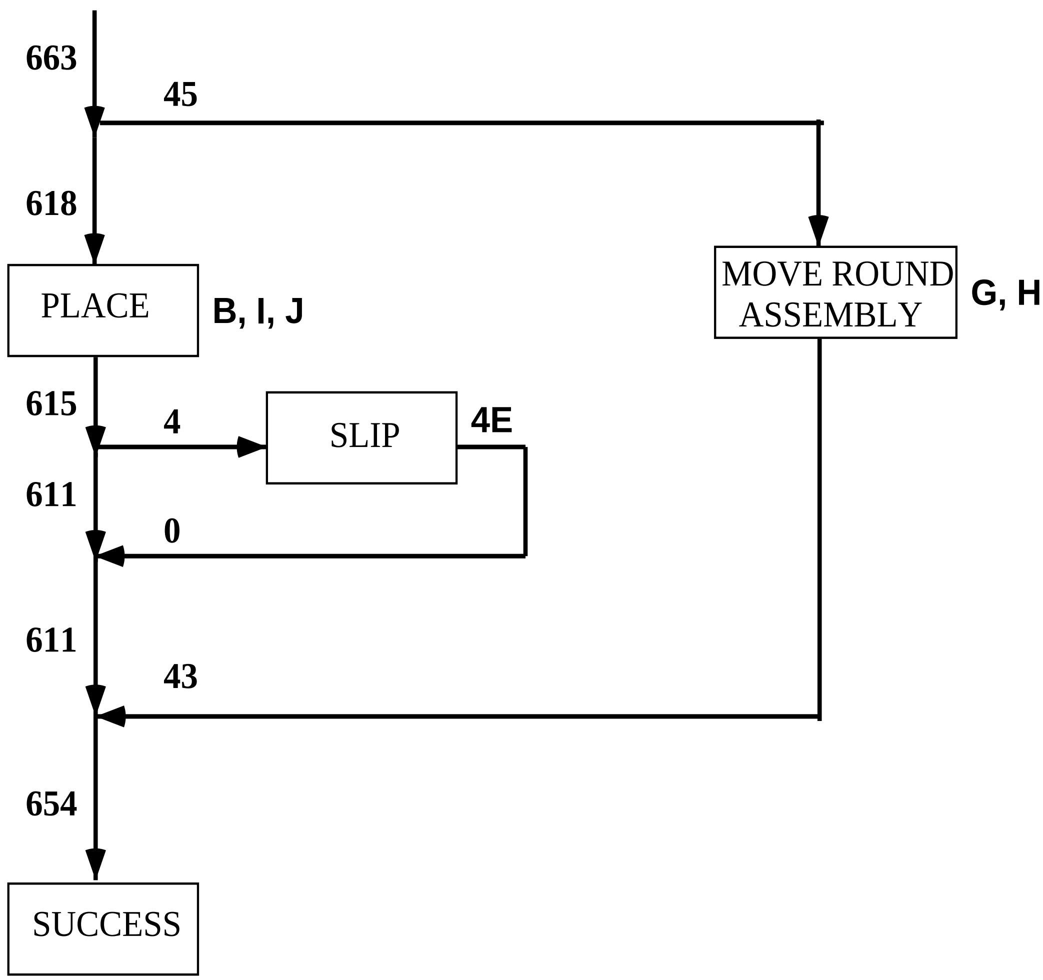


Figure 6–16: Experimental Runs using the complete set of Behavioural Modules (placing the part).

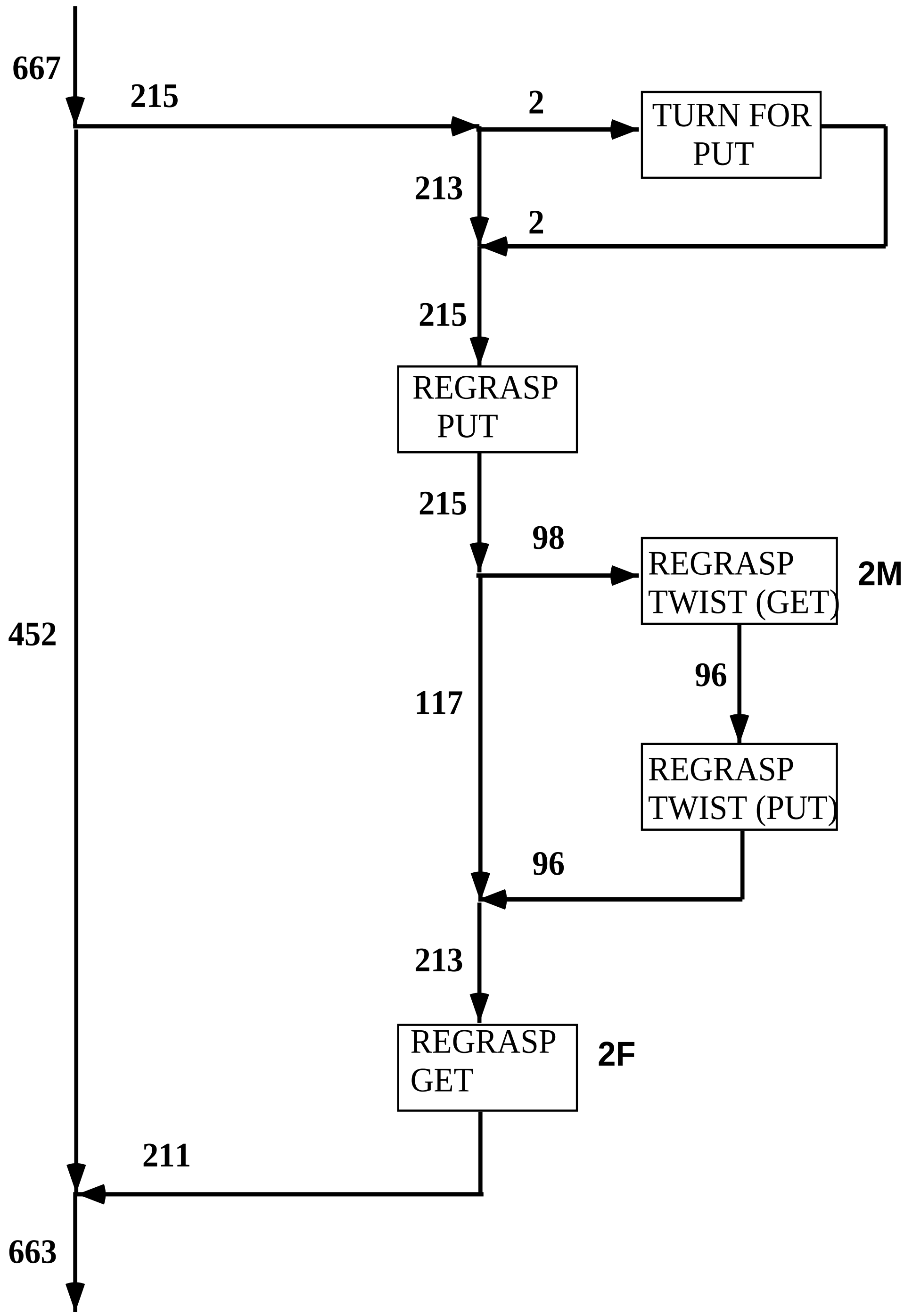


Figure 6–15: Experimental Runs using the complete set of Behavioural Modules (regrasp).

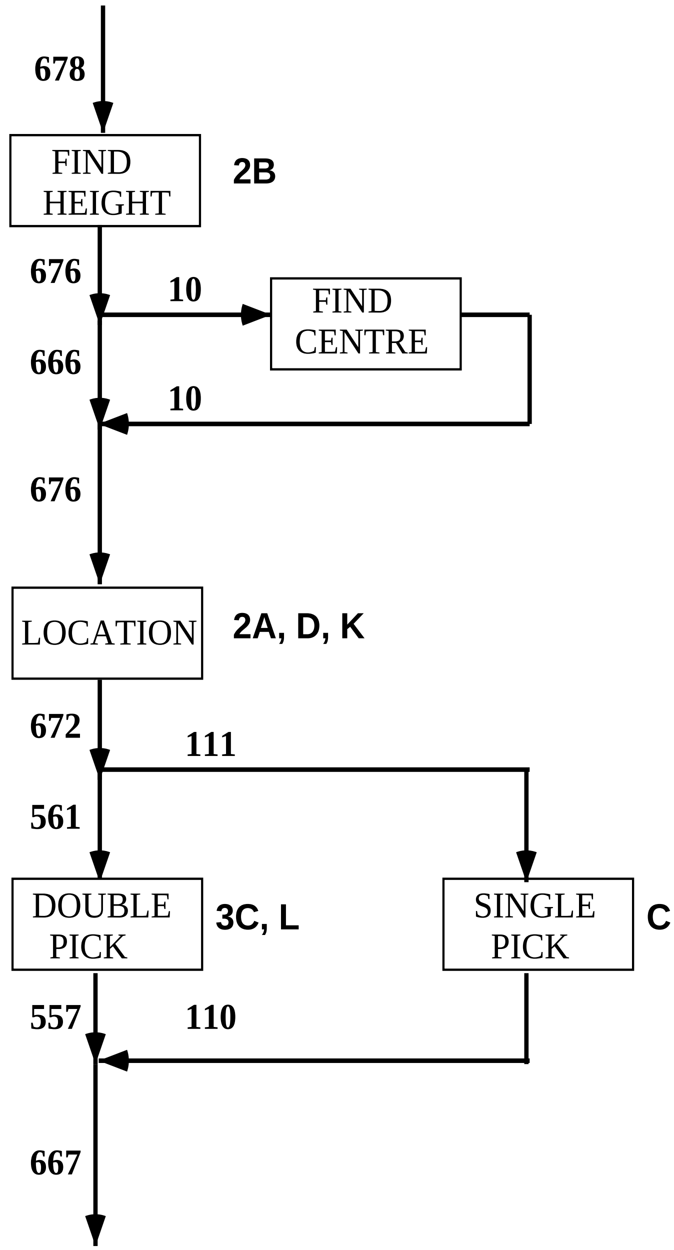


Figure 6–14: Experimental Runs using the complete set of Behavioural Modules (part acquisition).

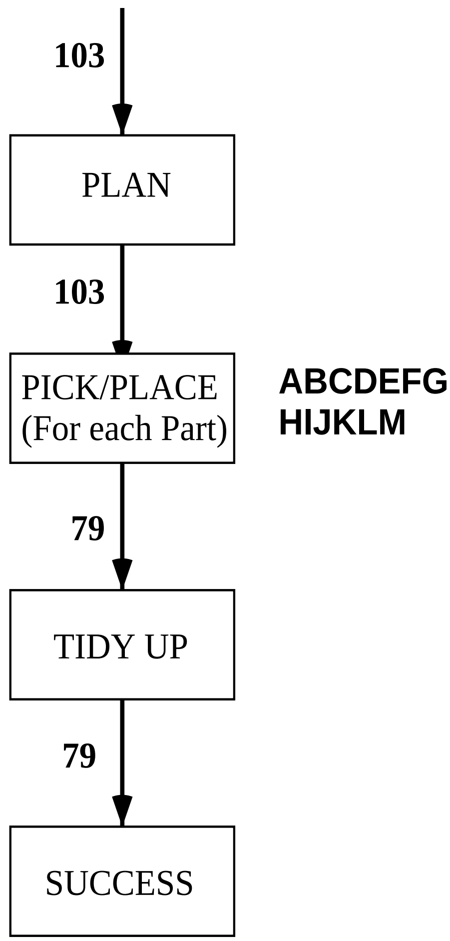


Figure 6–13: Experimental Runs using the complete set of Behavioural Modules.

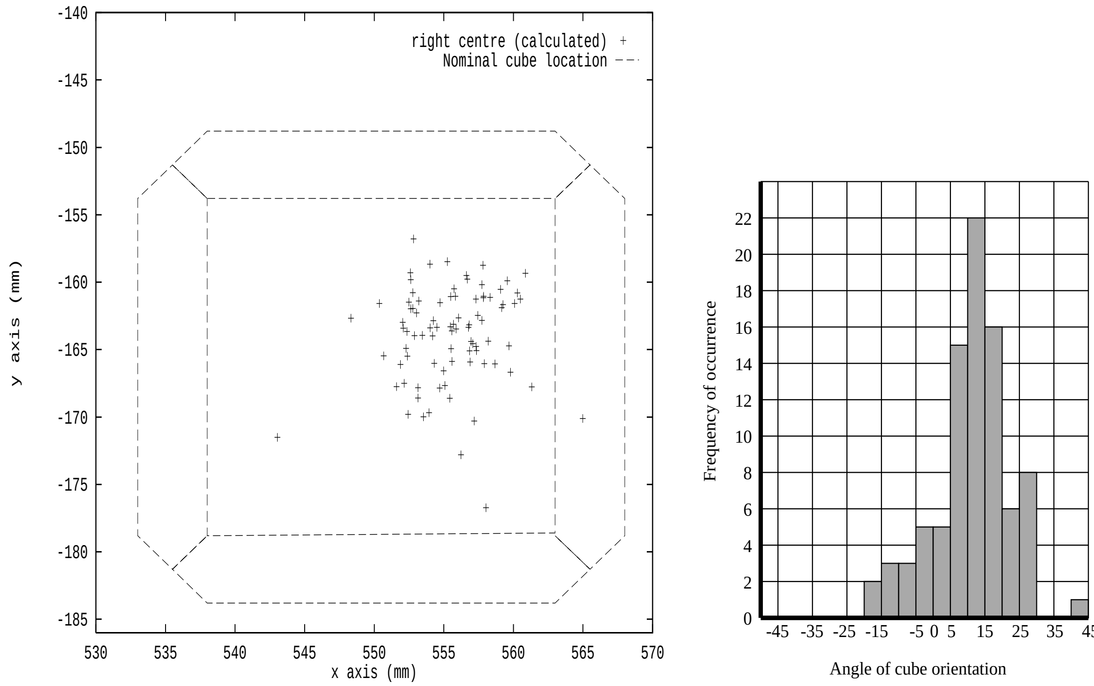


Figure 6–12: Calculated centres and orientation for the Soma part right.

Fig 6–13 shows the overall structure for each complete run of the assembly system, completing the assembly of the runs shown in Table 6.2. Each run involves initialisation, pick and place for each part, then a final Behavioural Module to tidy up the system and park the robot. Figs 6–14, 6–15, and 6–16 show the individual pick and place for each part in each of the above assembly runs (one run usually involved seven pick/place motions).

The diagrams do not show the complete set of Behavioural Modules used to perform the assembly task, but show the major ones on the Ideal Execution Path and the Alternative Execution Paths. From the diagrams, the Behavioural Modules on the Ideal Execution Path are, as for the Part Acquisition experiments, **Find Height**, **Location**, **Pick Double Snap**, and for the whole run of the system, the additional Behavioural Module, **Place** is also used. The Alternative Execution Paths available are **Find Centre** (entered 10 times), **Single Pick** (entered 111 times), the assorted Behavioural Modules used for Regrasping the part (**Regrasp** itself is entered 215 times, with 'turn for put' entered 2 times and **Regrasp Twist** entered 98 times), **Slip**

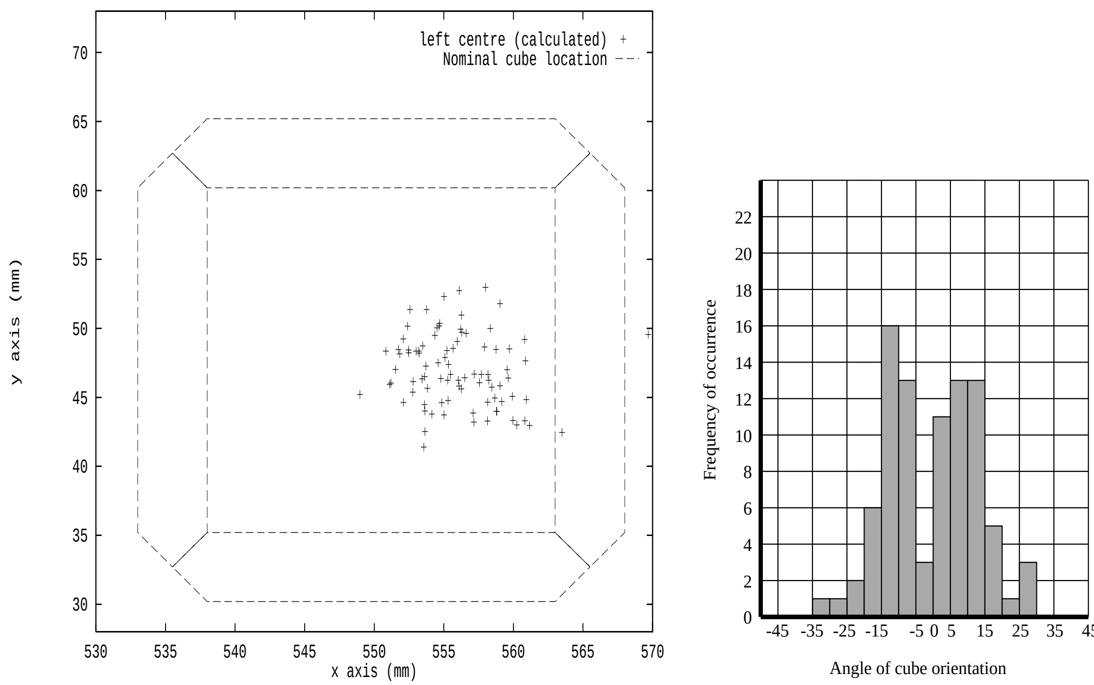


Figure 6–10: Calculated centres and orientation for the Soma part left.

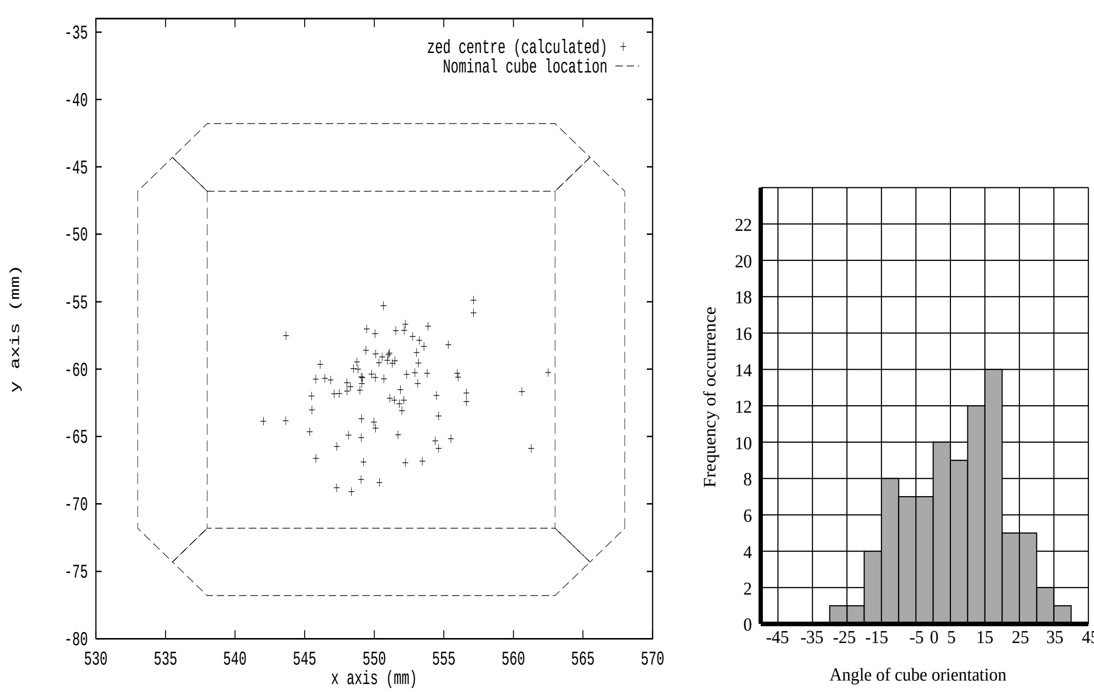


Figure 6–11: Calculated centres and orientation for the Soma part zed.

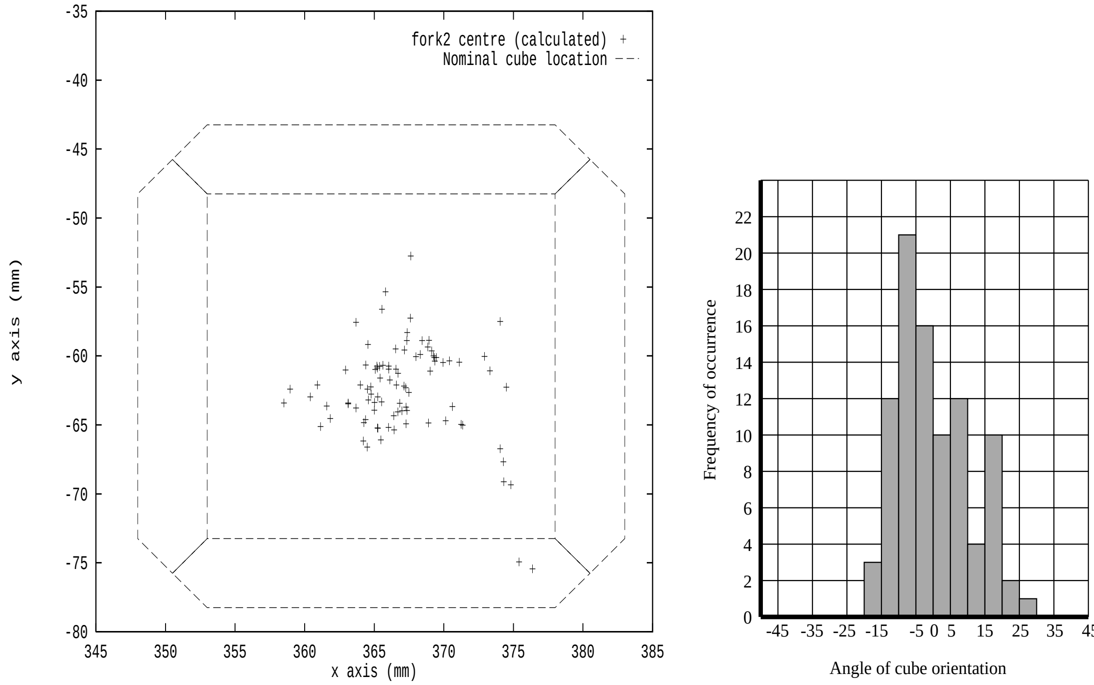


Figure 6–8: Calculated centres and orientation for the Soma part fork2.

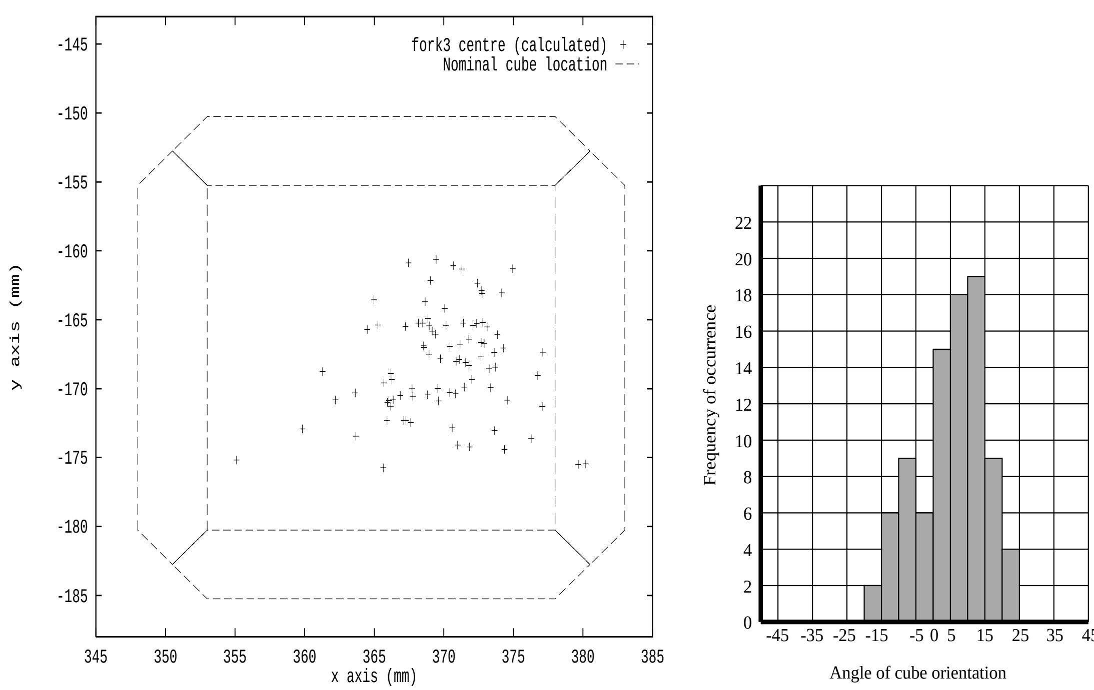


Figure 6–9: Calculated centres and orientation for the Soma part fork3.

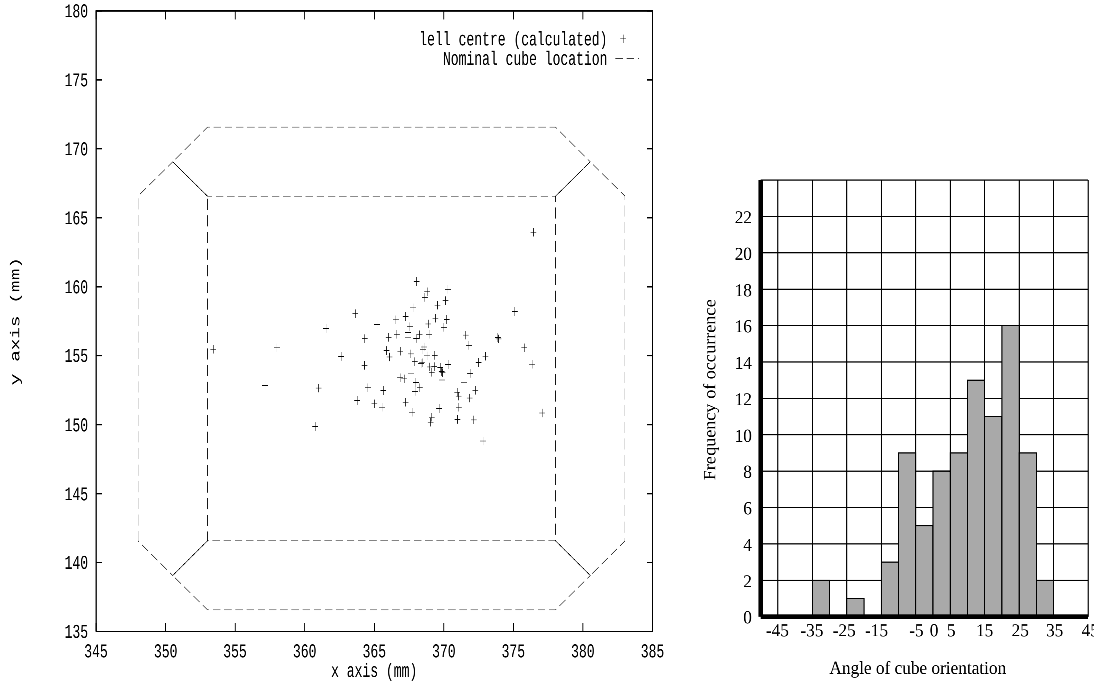


Figure 6–6: Calculated centres and orientation for the Soma part lell.

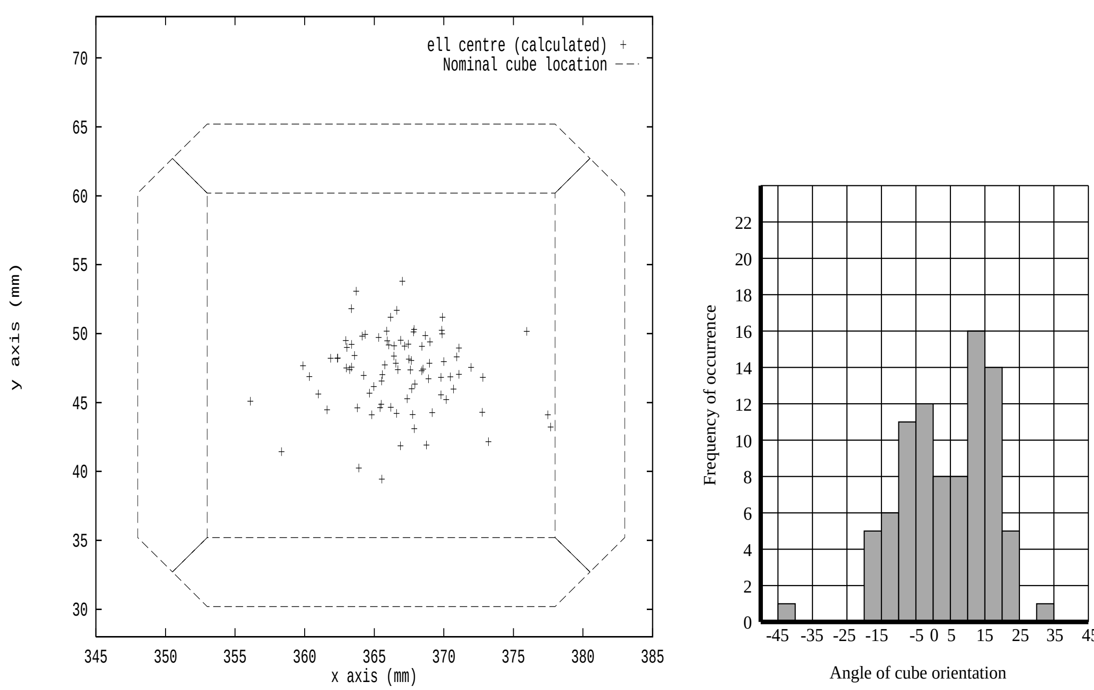


Figure 6–7: Calculated centres and orientation for the Soma part ell.

Run no.	Soma Set	Assembly
1–12	Soma-4	Cube333
13–16	Soma-4	Cube333
17–18	Soma-4	Cube333
19–40	Soma-4	Crystal
41–59	Soma-4	Cube333
60–79	Soma-4	Bathtub
80–99	Soma-4	Cube333
100–104	Soma-5a	Cube444

Table 6–2: Whole System assembly runs.

built. Most of the runs built different versions of the 3 x 3 x 3 cube which had different ordering of the parts and different regrasps required. The number of runs of each type of assembly was determined by the nature of the types of error situations which were occurring. Several sets of runs were curtailed when an error situation appeared at the same point in each run. In a developing system, alternative actions could be provided for these error situations, but as the system was not being developed further, nothing new was learned by continuing these runs.

Figs 6–6, 6–7, 6–8, 6–9, 6–10, 6–11, and 6–12 show the calculated centres worked out for each of the Soma parts, along with the associated range of calculated orientations. As for the Part Acquire runs, these calculations were worked out by the system. In this case, no actual values were taken. These diagrams give an idea of the variation present when the Soma parts were placed in the workcell by hand. The results were dealt with in seven diagrams as each part was nominally in one of seven taught locations.

The actual runs of the system are detailed in Appendix F, with the explanation for these given on page 201 of the Appendix. As with the results from the part acquisition experimental system, the results are summarised in a diagrammatic form, shown in Fig 6–13, Fig 6–14, Fig 6–15, and Fig 6–16.

the **Find Cube Top** level of the hierarchy, where another Behavioural Module could be inserted on the control path from the Exit States FH2, FH3, and FH4 to attempt to find the part using either different sensors, or by using a different strategy other than just a simple downward probe.

As well as implementing different actions to correct a problem, individual Behavioural Modules can be replaced with others incorporating more reliable strategies. For example, a different and more reliable search strategy could be implemented for finding the top of the cube in the **Find Centre** Behavioural Module.

6.2 Whole System Experiments

The second set of experimental runs involved the complete assembly system. These runs were set up to examine the reliability of the Behaviour-based system with the use of the Ideal Execution Path and Alternative Execution Paths. The strategies used by the Behavioural Modules are described in Chapter 5, Sections 5.1.2 and 5.1.3, and these perform the complete assembly.

The hardware and system set-up were as for the Part Acquisition experiments detailed above, with everything run from SICStus Prolog and using the fingers with six binary sensors (see Chapter 5, Fig 5–3 and Fig 5–4). The experiment involved picking up the Soma parts from their nominal taught locations in the workcell and placing them into the assembly. For these experiments, the Soma parts were just placed in the workcell by hand at the start of each run. The time taken to drop the parts from the gripper as for the Part Acquire experiment proved too long to be feasible with so many assembly runs. Placing the parts by hand provided a suitable amount of variation.

Most of the runs involved the Soma-4 set with its seven parts, but several runs were made with some Soma-5 parts to assemble a 4 x 4 x 4 cube. The Soma parts chosen to form the Soma-5a group are shown in Appendix D. Table 6.2 shows the different runs, the Soma parts used, and the assemblies

Module, **Find Centre** is entered, producing either the Exit State FC1 or FC2. These come out of **Find Cube Top** as either FCT1 or FCT2.

These tests have shown that, although the actual Behavioural Modules on the Ideal Execution Paths are not the best which could be designed, the addition of a useful Alternative Execution Path, determined from sensor information, can provide a more reliable system. Without the Alternative Execution Path, instead of the success rate being 91 out of 100, the successful pick up of the Soma part would have been reduced to 60 out of 100. Again, it must be realised that these runs were specifically designed so that the Soma part fell repeatedly to a position where it was likely that the Alternative Execution Path may be required. Only one run was lost during the deviation to the Alternative Execution path, which was due to the search strategy employed.

This testing does serve to illustrate a useful way of determining if the Behavioural Modules are appropriate for the task. If the Alternative Execution Path is used more often than initially anticipated by the designer of the system, or progressively becomes used more over a series of runs of the assembly, the workcell and the robot can be examined to determine the cause. The system, in the meantime, will continue its execution while the investigation is taking place as the Alternative Execution Path can currently deal with the increase in the variation. This could be due, for example, to a feeder which has shifted in alignment.

For each error situation found in the part acquisition system runs, the sensory coverage of the Behavioural Modules detected the problem and passed the Exit State parameter back up through the hierarchy of the system. At each level of the system, Alternative Execution Paths could have been used if required to prevent the system from falling into an error situation.

An example of this, (using the diagrams of the Behavioural Modules in Appendix B on page 182), could be applied to the error situation **E** where the Soma part was missed completely on the first downward probe of the sensors. Although detected at the very bottom level of the hierarchy by the **Check Sensor** Behavioural Module, an alternative action could be made available at

Module returns to this point, occasionally the sensor pushes the cube when trying to move horizontally.

B Incorrect orientation calculation — detected as the fingers move over the part to be picked up.

C Chamfer detected, but search strategy failed to find top of cube.

D Sensor slid off side of cube when moving back to the centre position to resume orientation finding strategy (due to slightly inaccurate tool coordinate frame change).

E Missed the cube completely on first downward probe.

Using the Diagrams in Appendix B which are too numerous to include in the main body of the text, the Exit States for the above error situations are as follows (beginning with the Behavioural Module lowest down the relevant hierarchy):

A CS2, FD5, LOC4, FLOC2, NP3.

B CS2, PDS4, NP2.

C CH2, CP2, HC2, FC2, FCT2, FLOC3, NP3.

D CS2, FD5, COP3, LOC3, FLOC2, NP3.

E CS2, FH4, FCT5, FLOC3, NP3.

The control path, again with reference to the diagrams in Appendix B, for the Alternative Execution Path which was used, sees the Behavioural Module **Find Cube Top** receiving the Exit State FH1 from the **Find Height** Behavioural Module. The returned height is compared with the floor height, then with the correct cube size, with the value agreeing with neither. When compared to a possible chamfer height, the values agree and the Behavioural

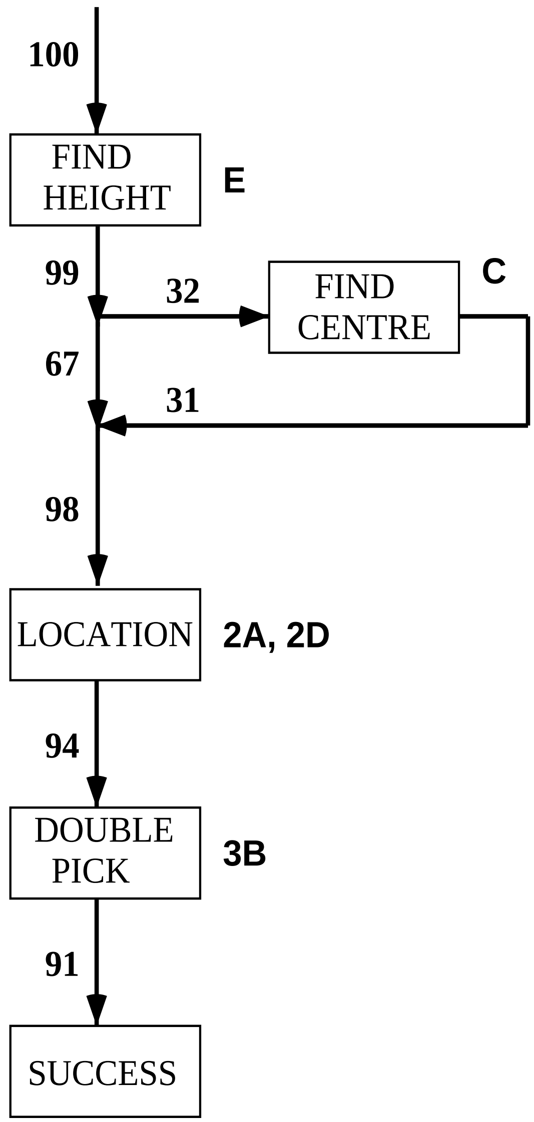


Figure 6–5: Experimental Runs using the Part Acquisition Behavioural Modules.

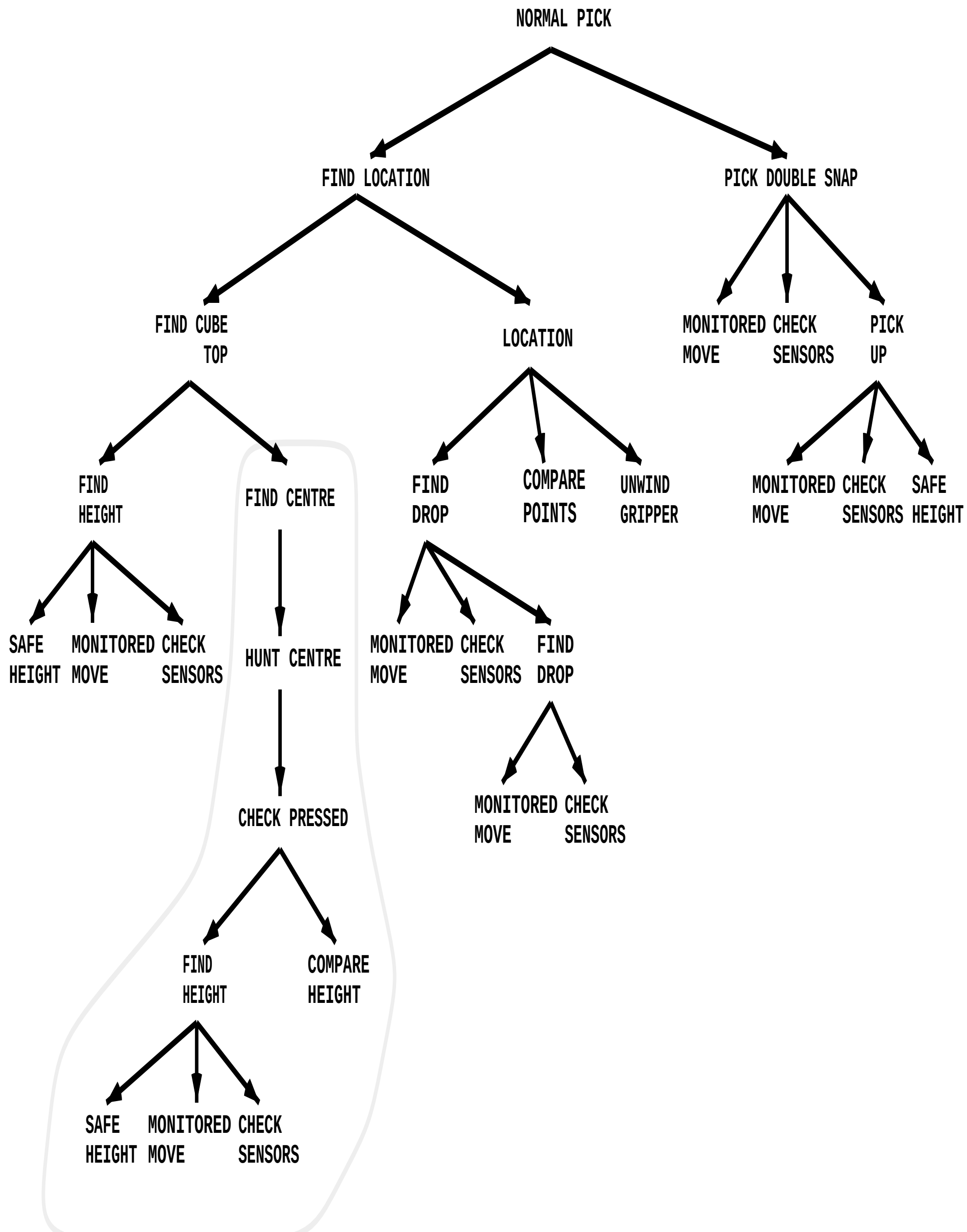


Figure 6–4: Pick Up Trial

structure showing possible calls of the Behavioural Modules in the lower levels. The internal structures of the Behavioural Modules are shown in full in the diagrams in Appendix B, Figs B–2 to B–18. The key for these diagrams can be found in Fig B–1 at the beginning of Appendix B.

Looking at the Behavioural Module hierarchy in Fig 6–4, the Ideal Execution Path for these runs involves the Behavioural Modules, **Normal Pick**, **Find Location**, **Find Cube Top**, **Find Height**, **Location** and **Pick Double Snap** (and the complete tree structures beneath these). The only Alternative Execution Path available for this task (enclosed in a dotted line) is the **Find Centre** Behavioural Module which is called if the positional variation of the Soma part is such that the initial downward probe of the sensor locates a chamfer on the top of the cube.

The actual runs of the system are detailed in Appendix E with an explanation given on page 196. A summary of these results is shown in diagrammatic form in Fig 6–5 which shows the Behavioural Modules, the number of runs which reached each part of the system, and a letter referring to the number of times each failure occurred. Out of the 100 runs of the system, 91 successfully completed the task even though, as has already been mentioned, the limits of the Behavioural Modules on the Ideal Execution Path were being tested. Of the 99 which made it through the **Find Height** Behavioural Module, 32 were immediately diverted to the **Find Centre** Behavioural Module on an Alternative Execution Path, indicating that a height corresponding to a chamfer had been found. 31 of these subsequently rejoined the Ideal Execution Path. The details of the failures are described next.

The actual errors which occurred and led to error situations are detailed in the following description, and where they occurred can be examined in both the summary diagram (Fig 6–5) and in the detailed runs in Appendix E.

A The **Find Height** Behavioural Module found a point right on the edge of the top just on the chamfer, and thus when the **Location** Behavioural

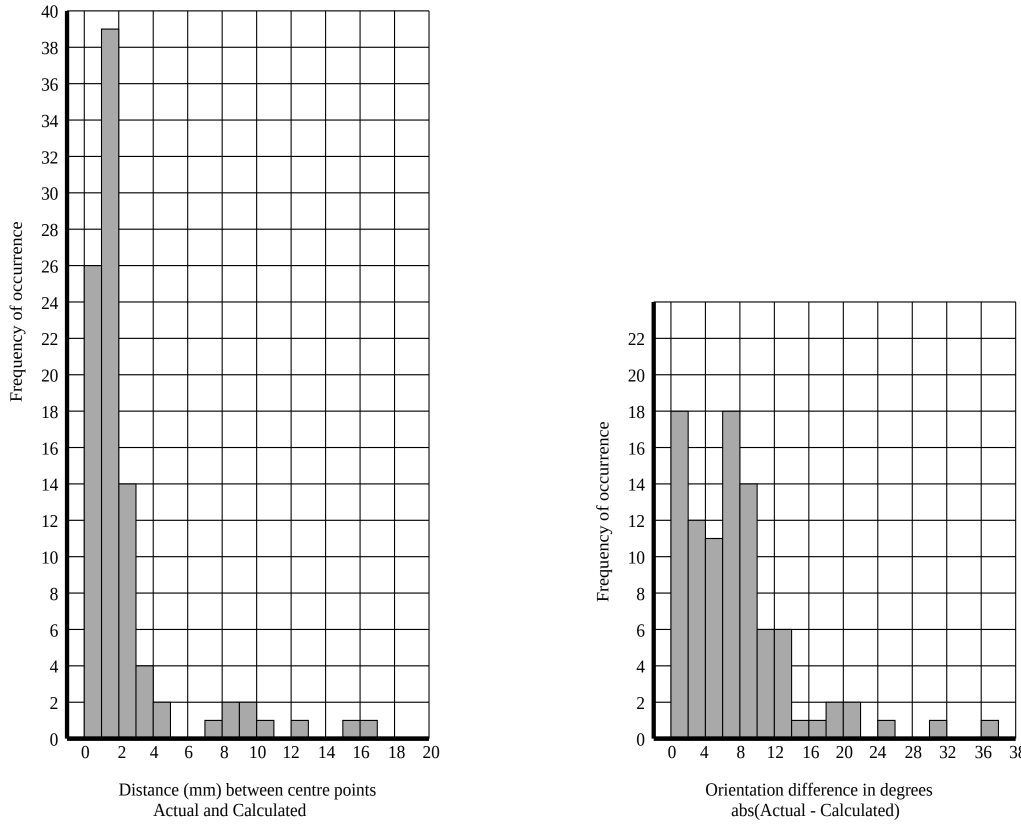


Figure 6–3: Difference between Actual and Calculated centres and orientations.

The diagrams in Fig 6–3 show histograms of the frequency of occurrence of the values of the difference between the actual and calculated centre points and the actual and calculated orientation values. The difference between the actual centres and the calculated centres is very small, considering the accuracy with which the actual positions could be determined (to within approximately $\pm$ 2mm). The orientations also show a similar close correspondence. As the gripper fingers, when fully open, can span approximately 50mm, with the cube size approximately 35mm, the difference between the actual and calculated values are within an error bound which can be dealt with by the system.

6.1.2 Pick-up results

The hierarchical structure of the Behavioural Modules used to execute the task of picking up the Soma parts can be seen in Fig 6–4, with each branch of the

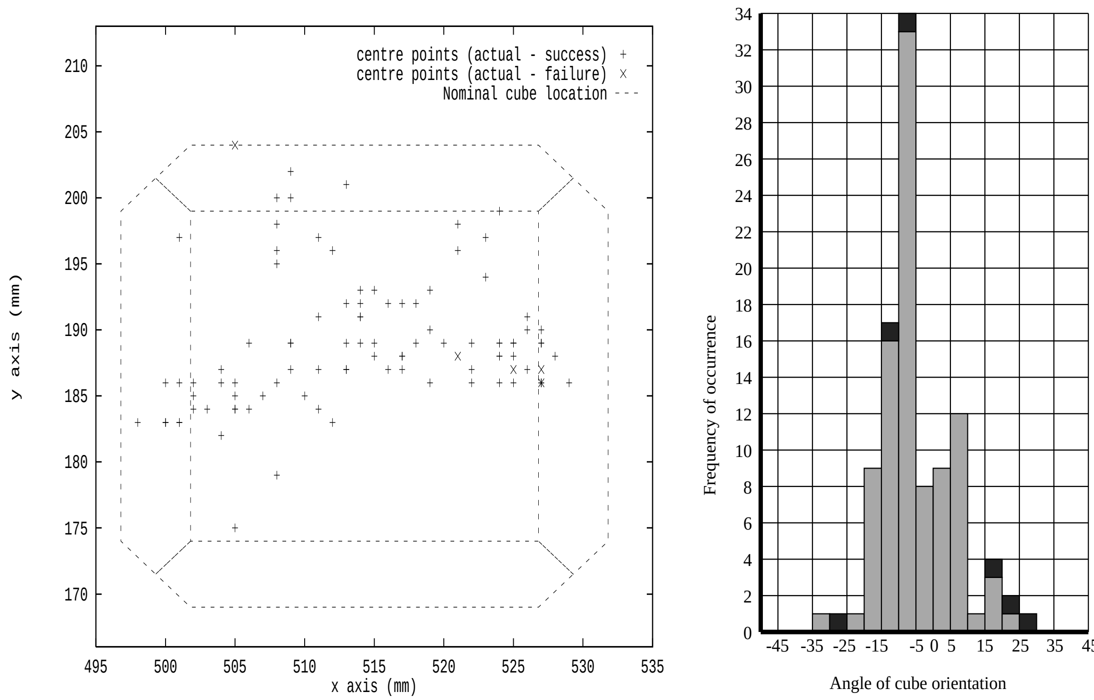


Figure 6–1: Actual centres and orientation of Soma cube for pick-up runs.

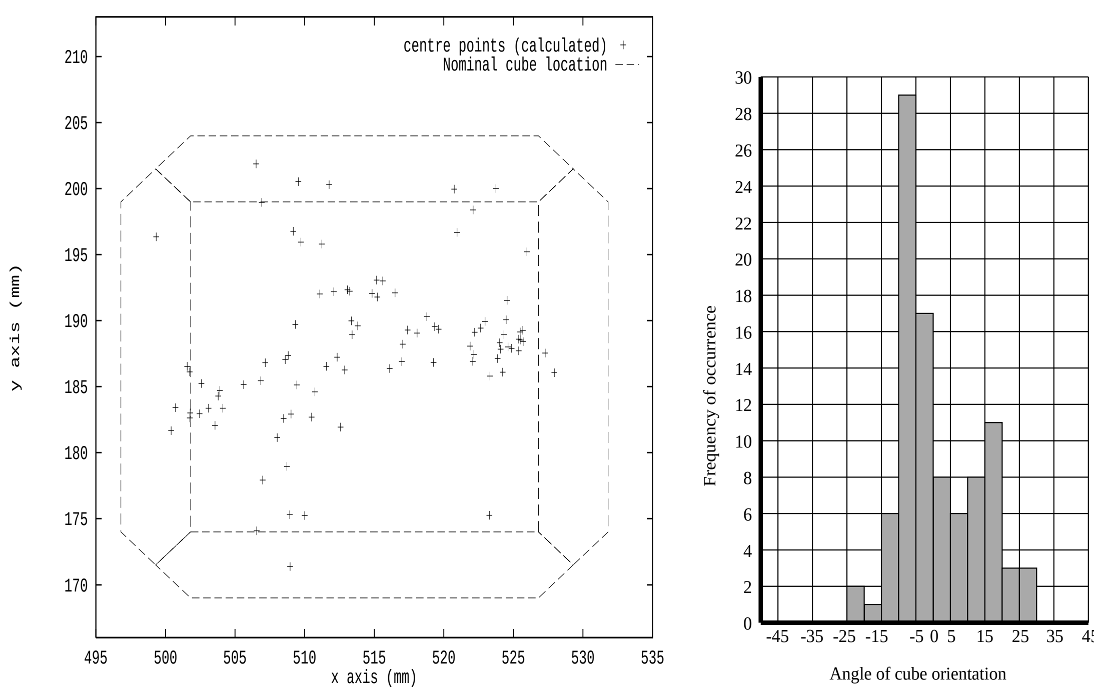


Figure 6–2: Calculated centres and orientation of Soma cube for pick-up runs.

Runs 76–100 were between these extremes. The choice of which parts to use in the experimental trials was arbitrary, with the only determining factors being that they were in a stable configuration with one single upstanding cube and they did not fall over into another configuration when dropped.

6.1.1 Variation

Before examining the details of the use of Alternative Execution Paths to deal with the failure of Behavioural Modules on the Ideal Execution Path, the difference between the actual position from which the part is acquired and the position calculated by the system are compared. This section will also give an idea of the variation which the Behavioural Modules can deal with in locating the Soma part. The actual centre points and orientations were measured from markings on graph paper while the system was being run, and the calculated centre points and orientations were taken from calculations performed by the system in its attempt to more accurately locate the part.

The diagrams in Fig 6–1 and 6–2 show the spread of actual and calculated centre points and orientations of the Soma part which have been gathered from the experimental runs. The two centre point diagrams show the actual and calculated centre positions, with the dotted outline of the chamfered cube showing the nominal position of the cube as expected by the system. These diagrams do not take into account the actual orientation of the cube at these centre points, thus leading to centre points where the probe sensor would appear, from the diagram, to land on the chamfer of the cube. This is not in fact the case as all the centre points shown are calculated from a point found by the initial downward probe landing on the cube top. The accompanying histograms show the orientation deviation from the nominal value of 0. The runs which failed before the centre point could be calculated are not shown in the calculated diagrams, but are indicated by a cross in the diagrams showing the actual centres and by darker shading in the histogram for the actual orientations.

Run no.	Part	Height(mm)
1–25	Right	20
26–50	Fork3	15
51–75	Fork3	22
76–100	Left	10

Table 6–1: Soma part pick up runs.

The hardware for these experiments is described in Section 5.1.1 (gripper fingers from Experimental System 2, and the working environment from Experimental System 3), with the complete system run from SICStus Prolog. As the system ran, the Exit States and control directions were stored where they could be examined to analyse the system. This information proved useful in determining the control flow through the Behavioural Modules and in identifying any problem areas.

The experiment involved picking up Soma parts which had a large degree of positional variation. For these runs, the variation was achieved by dropping the part from the robot gripper, with the part bouncing on the table dependent on its shape. This kind of delivery was used in these experiments as the mechanical repeatability gave sets of runs where the part landed at the edge of the competence of the Behavioural Modules. The part was picked up using the strategies and Behavioural Modules detailed in Chapter 5, then dropped again with the actual position of the dropped part noted (to within approximately 2mm). An approximation to this position is calculated by the system during the Behavioural Module **Location**, and this is compared with the actual position in Section 6.1.1.

Table 6.1 shows which Soma parts were used (see Fig 3–3 for the names and shapes of the Soma parts), the run numbers, and the height from which the parts were dropped. Runs 1–25 and 51–75 resulted in the Soma part landing near the edge of the capability of the Behavioural Modules on the Ideal Execution Path, with runs 26–50 tending to be well within their capability.

Chapter 6

Experimental Results

The experimental system described in Chapter 5 was developed from the original SOMASS system to investigate increasing the reliability of such a Behaviour-based system. To do this, sensors were introduced, along with the concept of Exit States, the Ideal Execution Path, and Alternative Execution Paths, which are described in Chapter 4.

This chapter examines some results obtained from two types of experimental system runs. The first of these, in Section 6.1, test the Behavioural Modules designed to acquire the Soma part while the second set, described in Section 6.2, involve complete runs of the whole system.

6.1 Part Acquisition Experiments

The purpose of this set of experimental runs was to test the reliability of the system in picking up Soma parts. The Behavioural Modules which were designed for the system were not in themselves the most reliable which could be built for the task, following the relaxation of the idea that Behavioural Modules must be very reliable task-achievers, but instead provided an adequate competence. This allowed the system to be tested to see how well the Ideal Execution Path and Alternative Execution Paths could improve the reliability of the system.

Behavioural Module hierarchy builds up from lower level Behavioural Modules. These show how the sensory and local data are used as decision points to determine the Exit State value.

Sections 5.4 and 5.5 examine the control paths which have been chosen for this system and how these are identified from Exit State values and some local decisions. Behavioural Modules on Alternative Execution Paths can either deal with problems which have occurred during the execution of a Behavioural Module on the Ideal Execution Path, such as dealing with a slipped part in the gripper, or may have completely different competences more useful to deal with the actual circumstances in the workcell, as can be seen by the example of the **Move Round Assembly** Behavioural Module which can cope with assemblies of a greater height than the normal **Place** Behavioural Module. Several of these Alternative Execution Paths are described in Section 5.5.

the centre of the cube along the other. On picking up the Soma part, the lower two sensors between the gripper fingers inform the system when the gripper moves over the part, as seen in Fig 5–14, providing a more accurate gripping position and allowing the sensors to monitor slippage in the gripper.

When using the binary sensors to stop the robot and report the location back to the system, the speed at which the robot is run provides different locations for different speeds due to the time it takes for the sensor to trigger and halt the robot. This means that the interpretation put on these results by the system are speed dependent. To keep the bounds of the uncertainty within a range that the system can still deal with, the system is run at the same speed whenever reported guarded motion locations are required.

The lightbeam sensors incorporated into this assembly are simple on/off binary sensors, and so the programmer does not have the same thresholding problems which are found in force and vision sensing. The uncertainty introduced in some of the sensing strategies is either within the bounds that the system can cope with, can be further reduced by strategies such as the example of running the robot at a constant speed, or the Behavioural Module can be accepted as being susceptible to failure and carefully monitored to see if the strategies have correctly completed their task.

5.6.3 Exit States, the Ideal Execution Path, and Alternative Execution Paths

The experimental system described in this chapter uses Exit States, the Ideal Execution Path, and Alternative Execution Paths to increase the reliability of the assembly. This chapter has outlined the Behavioural Modules used in the experimental system. Sections 5.3, 5.4, and 5.5 examine some of the Behavioural Modules in more detail with respect to the concepts behind Exit States, the Ideal Execution Path, and Alternative Execution Paths.

Section 5.3 examines the Exit State values of some of the Behavioural Modules in the experimental system, and how these relate to each other as the

- Variation and uncertainty handling strategies.
- The use of Exit States and local information to determine the Ideal Execution Path and Alternative Execution Paths.

5.6.1 Planner strategies

As with the SOMASS system described in Chapter 3, certain strategies are enforced by the planner with the aim of providing a more reliable assembly. The planner enforces vertically downward acquisition of the parts by providing a gripper orientation to this effect, and similarly for part placement (although this constraint is removed when placing and picking the part up from the regrasp table). The planning of the assembly also ensures that each cube in the Soma part to be inserted into the assembly is supported either by the workcell floor or by an already assembled part. All rotations provided by the planner are such that they can be performed by the 5 degree of freedom robot which was used for the assemblies.

5.6.2 Variation and Uncertainty Handling Strategies

Several strategies are used in the course of the assembly execution to reduce the amount of variation in the workcell. The position and orientation of the Soma parts are more accurately determined by using the probe in the fingertip of the gripper to investigate the appropriate area, as described in Section 5.1.2. This does not reduce the variation present in the part position, but allows the position to be more accurately determined. The system must now also take into account any uncertainty introduced by the sensory measurements if it is still to perform reliably. A double snap of the gripper, as shown in Fig 3–16, is used to centre the part in the gripper if a single upright cube is available for grasping, physically reducing the variation present in the part position. If this is not the case, a single snap helps to reduce the variation along one axis, and running the probe across the top of the cube, as seen in Fig 5–18, helps to find

be rotated through 180 degrees and then placed down as described in Section 5.1.3. This rotation forms another minor Alternative Execution Path, depending on the information from the robot controller checking if the robot can physically reach a position.

If the robot cannot reach the cube to be grasped, in the orientation that it must use, another Alternative Execution Path is used. This time, the alternative involves a Behavioural Module, **Regrasp Twist**, which performs a pick up, rotation and place (also described in Section 5.1.3). If this Behavioural Module returns the Exit State RT1 showing that the rotation has been successfully achieved, the system can go on to regrasp the correct cube and the control returns to the calling Behavioural Module.

Slip

The **Slip** Behavioural Module is only used if during the transportation of the Soma part, the system detects that it has slipped in the gripper. An attempt is then made to pick up the part again as described in Section 5.1.3.

Move Round Assembly

This Alternative Execution Path is determined from data from the planner and performs a type of collision avoidance, since the main object of this Behavioural Module is to avoid the already completed assembly.

5.6 Summary of the Experimental System Reliability

The experimental system uses three different methods to deal with the variation and uncertainty found in the workcell. These are as follows:

- Planner enforced strategies.

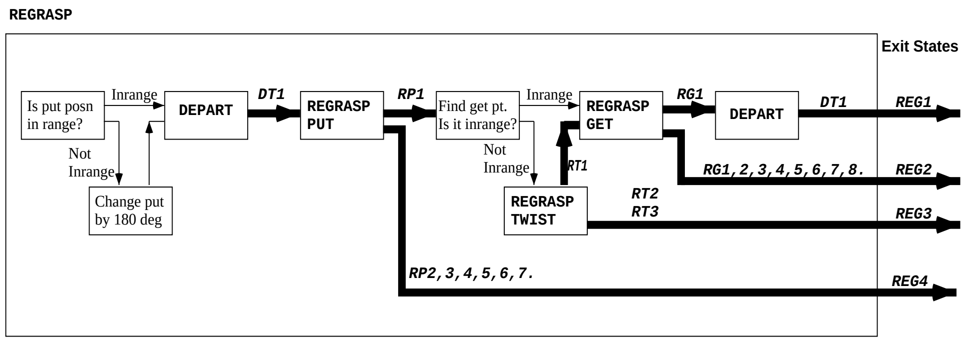


Figure 5–25: Regrasp Behavioural Module.

Behavioural Module (see Fig 5–23) and tries points a small distance along the ± x axes and ± y robot axes until the top is found as illustrated in Fig 5–16.

Single Pick

The diversion from the Ideal Execution Path of the normal pick up Behavioural Module, **Pick Double Snap**, is determined by information from the planner. If any cubes are joined to the one to be gripped at the same height as that cube, the strategy detailed in Section 5.1.2 is used. Information about cubes joined at the same height as the gripped cube could be easily discovered by the probe sensors in the fingers, but with the planner interfaced at the Prolog level, and the information about the parts available there, it proved less time consuming to consult this to determine if the Alternative Execution Path should be followed.

Regrasp

The **Regrasp** Behavioural Module is only entered if information, again from the planner, states that the part cannot be taken from the initial starting location to its place in the final assembly with only rotation around the z robot axis.

Once in the **Regrasp** Behavioural Module (Fig 5–25), if the robot cannot physically place the part down on the regrasp table, then the part must

within a Behavioural Module), an Alternative Execution Path may be followed to make use of this information and to try to restore the system to the Ideal Execution Path. It is useful to know how a Behavioural Module exited as, if the Behavioural Module is well designed, this can tell the system exactly what happened within it, allowing the correct choice of an Alternative Execution Path.

In the original SOMASS system the Ideal Execution Path consisted solely of the pick and place Behavioural Modules. The only Alternative Execution Paths available were entered using information from the planner. These included the regrasp strategy and the pick up Behavioural Module for the 'zed' part. All other actions took place whether the assembly was being performed correctly or not. If the variation in the workcell proved too great for the competence of the Behavioural Modules, the robot would still continue along the Ideal Execution Path although there was no way that the assembly could be correctly completed in this way.

Several Alternative Execution Paths were designed for the experimental system for this thesis. The Ideal Execution Path has already been described in Section 5.4. This section will describe some of the major Alternative Execution Paths which were specifically designed to operate in this system, giving examples from the Behavioural Modules already described in the previous sections.

Find Cube Top

As has already been described in Section 5.3, this Behavioural Module (Fig 5–24) locates a point on the top of the cube to be gripped of a Soma shape. If a point is detected which corresponds with the height of a chamfer on the cube, an Alternative Execution Path is used.

This particular Alternative Execution Path uses the Behavioural Module **Find Centre** to locate a point on the top of the cube and bring the system back to the Ideal Execution Path. If a chamfer has been located, the top of the cube must be somewhere near, but it is not known in which direction. The **Find Centre** Behavioural Module uses the **Find Height**

The Ideal Execution Path for the experimental system begins at the top level of the hierarchy, as described in Section 5.1.2, with calls to the Behavioural Modules to acquire the parts, and to place them in the assembly. Both of these are broken down into other Behavioural Modules as the hierarchy is traversed.

The example Behavioural Module, **Find Cube Top** (See Fig 5–24), is part of the Ideal Execution Path for the system. The path progresses through the Behavioural Module **Find Height**, with Exit State FH1, the 'Compare with floor height' local module, with control information that floor height has not been found, and ends up in the 'Compare Cube size' local module with the correct cube size found. This leads to an Exit State of FCT1 from the **Find Cube Top** Behavioural Module.

At the next level down, the **Find Height** Behavioural Module also lies on the Ideal Execution Path, due to it being called from the Behavioural Module above already on the Ideal Execution Path. The Ideal Execution Path through the **Find Height** Behavioural Module follows **Safe Height** with Exit State SH1, **Monitored Move** with Exit State MM1, **Check Sensors** with Exit State CS1, **Monitored Move** with Exit State MM1 and finally **Check Sensors** with Exit State CS1, ending up with Exit State FH1 from this Behavioural Module. Although two paths end with Exit State FH1, only the path just described forms the Ideal Execution Path for this call of the Behavioural Module.

5.5 Alternative Execution Path

In developing the complete system, several opportunities occurred to make the most of the sensor and planner information available to allow the control of a Behavioural Module to follow Alternative Execution Paths. The above Ideal Execution Path is followed if the Behavioural Modules can deal with the variation and uncertainty in the workcell. If they cannot (indicated by the context of the Exit State from a Behavioural Module or by control information

position corresponds to the workcell floor. If it does, then Exit State FCT4 is returned. If not, the returned position is then compared with the nominal expected height of the cube, and if there is a match, Exit State FCT1 will be returned.

If the height does not match with the expected top of cube height, the height is then compared with a possible chamfer height. If the height matches with this, a Behavioural Module on the Alternative Execution Path, **Find Centre** (described in Section 5.1.3) is entered to move the robot to the top of the cube. Corresponding Exit States will be returned depending on how the **Find Centre** Behavioural Module is exited, producing FCT1 if the top has successfully been found, and FCT2 if not. Diagrams showing the **Find Centre** Behavioural Module, and the Behavioural Modules it calls, are given in Appendix B. If the height, on the other hand, does not correspond to the height of the chamfer, the Exit State FCT3 is returned.

5.4 Ideal Execution Path

At each level of the Behavioural Module hierarchy, the Ideal Execution Path can be traced by the Exit States from lower level Behavioural Modules called from this level, and by other control decisions which are made from information available to the system at this point. Exit States define a certain way of exiting a Behavioural Module and although, within a Behavioural Module the same Exit State may be reached by several different paths, from the outside of the Behavioural Module only the result is seen. An example of this is in the **Find Height** Behavioural Module, see Fig 5–23, where Exit State FH1 can be arrived at by two different routes. Therefore, to determine the Ideal Execution Path through a system which has been built, the path is followed down through the hierarchy and is determined by Exit States and local decisions.

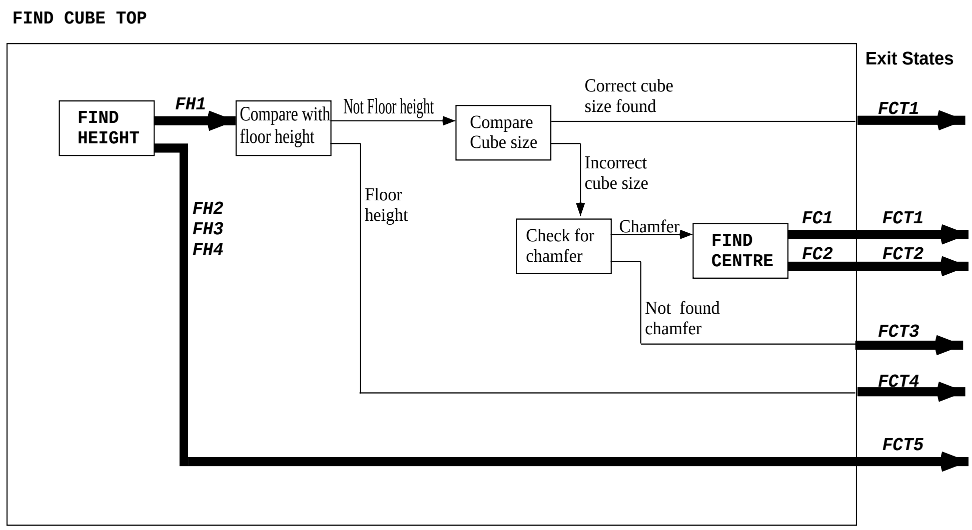


Figure 5–24: Find Cube Top Behavioural Module.

If probe A does not trigger, probe B is then checked. If this has not been triggered, Exit State FH2 is returned, signifying that no object has been found. If it does trigger, Exit State FH3 is returned, signifying that probe B has found some height.

The Exit State value FH1 corresponds to a height being found by the correct probe. A parameter containing the robot's position is also returned.

Find Cube Top

In moving up one more level, this Behavioural Module, **Find Cube Top** uses the previous module, **Find Height**. Designed to locate the top of a Soma shape in the workcell, when a nominal location of the part is known, this module uses the **Find Height** module, then determines if the height returned is useful. Fig 5–24 shows a diagram of the Behavioural Module.

The **Find Height** module can return one of the Exit States, FH1, FH2, FH3, and FH4. Exit States FH2, FH3, and FH4 from **Find Height** result in Exit State FCT5 from the **Find Cube Top** Behavioural Module.

If FH1 has been returned, signifying a height has been found, the parameter containing the robot's position is examined to determine if the

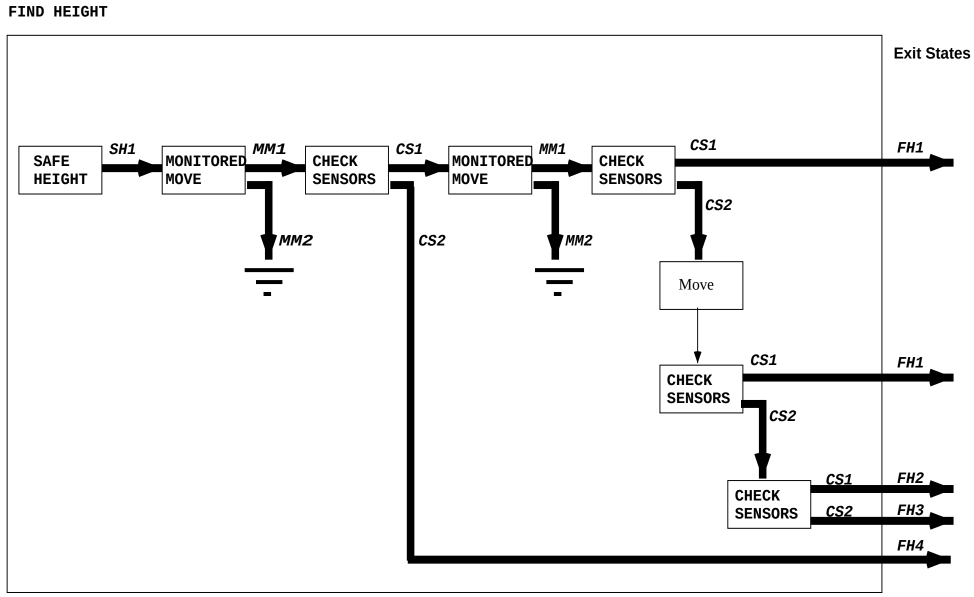


Figure 5–23: Find Height Behavioural Module.

Height. The state of the sensors is now compared with their expected states, to determine if any sensors have been triggered on the way to the safe position. If this has happened, Exit State FH4 is returned and the Behavioural Module exited immediately.

If the sensors are in the expected state, the Adept is then commanded to move vertically downward towards the workcell floor. The robot will either be stopped by the sensors triggering, having touched something, or by reaching the position on the floor. The height finding probe, for this example, probe A, is checked to see if it has activated. If it has, the Behavioural Module exits with FH1.

If the probe which has been designate to find the height, probe A, has not been triggered, the robot is commanded to move down a very small amount, to check if perhaps the other probe, probe B, has hit the object first. If probe A now triggers, a valid height has been found, and, again, Exit State FH1 is returned. This may happen if probe A lands over a chamfer.

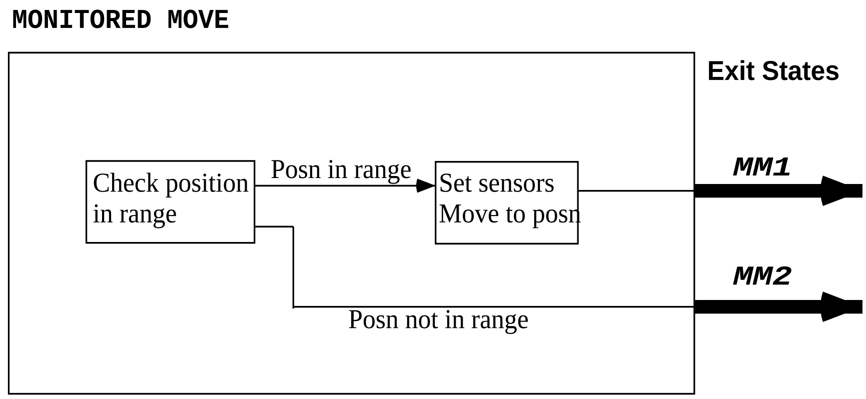


Figure 5–22: Monitored Move Behavioural Module.

The first Exit State, MM1, signifies that the Adept has either successfully moved from its current position to the named position, carried in a parameter, or it has been halted along the way when one of the monitored binary sensors successfully triggered. Both of these results are regarded as the same Exit State for this particular Behavioural Module. The only other Exit State, MM2, is one which signifies that the position which the Adept was told to move to could not be reached by the robot. This can easily occur if the position has been calculated.

Find Height

As a more complex Behavioural Module, **Find Height** has more Exit States than **Monitored Move**. The origin of the Exit States can be determined by examining the component sections of the Behavioural Module in Fig 5–23.

The Behavioural Module begins by calling another Behavioural Module, **Safeheight** which can be seen in Fig 5–21. This Behavioural Module always returns Exit State SH1 and returns a position which is in reach of the Adept at a safeheight above a designated point on the floor of the workcell. This means that, by default, the two sensor **Monitored Move** to this position will produce the successful Exit State, MM1. The Exit State MM2, which is related to the position to be moved to being out of range, is not needed and is therefore not returned in any form from **Find**

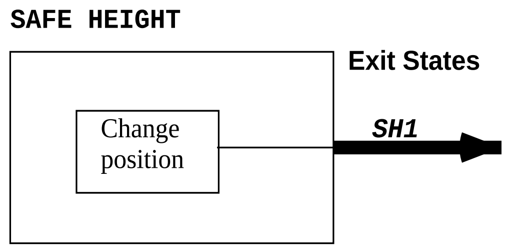


Figure 5–21: Safe Height Behavioural Module.

5.3 Exit States

This section expands some of the previous examples to include the possible Exit States from the Behavioural Modules. As has been previously stated (see Section 4.3), Exit States are the values returned from a Behavioural Module to inform the system of how it has exited. Behavioural Modules can be designed to return these Exit States, which can then be used by the control hierarchy to determine if a Behavioural Module has successfully completed its task in the context of its calling, or whether something has failed. It is important that the sensory coverage is adequate and appropriate for the task being performed.

Safe Height

This Behavioural Module is an example of the simplest Exit State description possible. There exists only one Exit State from the Behavioural Module, SH1, which reflects the fact that the only task this module performs is to return a position which is within reach of the Adept at a safe height above a designated point on the floor of the workcell. The diagram in Fig 5–21 shows this simple Behavioural Module. The key for the diagram can be found in Section 4.2 on page 88.

Monitored Move

The Behavioural Module **Monitored Move**, which is described in Section 5.2, has two very simple Exit States which are shown in Fig 5–22.

Monitored Move. This gave an accurate enough location to lower the gripper over the object without catching the top.

Example 5 — Pick Double Snap

When dealing with grasping Soma shapes which have a single upstanding cube designated as the cube to be gripped (which is true of all parts except zed in the Soma-4 set), a Behavioural Module similar to the one used for the same purpose in the SOMASS system was used.

Having been provided with the location of the part as a parameter, the robot moves down over the part, and performs two snaps of the gripper, at 90 degrees to each other. This ensures that the part is centred in the gripper, which is useful for variation reduction. As has already been mentioned in Section 5.1.2, the additional strategy introduced in the system with sensors involved moving down over the cube to be gripped until the bottom two lightbeams located between the fingers were obscured. The robot was then halted and the fingers closed. In this way, the sensors reported whether the gripper was holding the cube, and could tell if it slipped. All the movements in this Behavioural Module were performed by the Behavioural Module, **Monitored Move**.

These are all examples of Behavioural Modules which were designed, implemented and tested individually as they were developed, checking that their Exit States (some of which are examined in the next section) were relevant to the task of the Behavioural Module. In this way, when combined, the competence of the underlying Behavioural Modules could be relied on to perform to a certain specification.

other sensor is triggered. This means that the height found will be the height under the designated probe. The two sensor **Monitored Move** Behavioural Module is used on all movements of the robot in the **Find Height** Behavioural Module.

Example 3 — Find Drop

Again, stopping the robot using a binary sensor transition plays an important part in this particular Behavioural Module. Of particular use in the Soma domain where the shapes are cubes, once the height of an object is found using the **Find Height** Behavioural Module, the robot can then move horizontally until the probe sensor is released to find the vertical edge of the cube. This has proved useful for finding locations and orientations of the cubes. Again, this uses the **Monitored Move** for the guarded motion to find the drop point. These drop points can be found using Soma cubes with chamfers.

Example 4 — Location

Initially, the problem of picking up the Soma shapes was that of locating the parts accurately enough in the workcell to be picked up without the gripper catching on the upper surface of the cube on the way down to grasp it. Variation in position and orientation could be so great that just bringing the gripper down to straddle the part would often result in the gripper becoming stuck on the top of the object. The sensorless SOMASS system relied on sweeping the parts into place to reduce this variation and allow the parts to be correctly grasped, but this required room around the parts in the workcell for the sweeping motions. As more parts were introduced, a problem arose concerning the space in the workcell, and so a different orientation finding strategy was devised which used binary sensors.

A reliable method for finding the orientation was devised, as detailed in Section 5.1.2, mainly using the Behavioural Modules **Find Drop** and

module, while constantly monitoring the binary sensors in the fingers for particular state changes which are also detailed as parameters. If one of the specified sensor transitions is detected, the Adept motion is halted immediately.

To be useful in many situations, a Behavioural Module must be sufficiently general; in this case, this is achieved by parameterisation. As has been explained in Section 5.1.1, the binary sensors can be constantly monitored to detect changes in state. By setting parameters to detect particular changes, the four sensors located between the fingers can detect the presence or absence of an object in the gripper, and also whether an object has slipped in the gripper.

All the sensors can be used to perform guarded motions. An example of this is to find the vertical height of an object by moving downward until the probe sensor is triggered, as is used in the Behavioural Module **Find Height**. The robot controller can be queried for the current robot position if that position is required on completing a guarded move. If the Behavioural Module is being used only to stop the robot, the actual stopping location may not be important. This Behavioural Module can be used for many different purposes, from guarded motions to checking if an object has dropped from the gripper as it moves.

Example 2 — Find Height

Using a version of the previous Behavioural Module, **Monitored Move**, which only monitors the two probes, this Behavioural Module is designed to return a position, representative of the vertical distance between the workcell surface and the current gripper position, thus determining a height. The Adept moves to a safe height over a point which is provided as a parameter, then, monitoring the two sensors, moves vertically downward until either a sensor is triggered to halt the robot, or the given point is reached. In this particular module, only one of the probes is designated as able to find a height, with an error being announced if the

There are many other minor Alternative Execution Paths in the assembly system, all activated by sensory information from the real world, or by information from the planner translated into real world terms.

5.2 Behavioural Modules

As the hardware and software in the experimental assembly system developed, so too did the design and implementation of the Behavioural Modules. The main changes described above involved inventing new and more robust strategies, and also using the sensors in ways better designed for the task of assembling Soma models. The following examples outline in more detail some of the Behavioural Modules which were developed and used for the experimental trials shown in Section 6.1, to give an idea of the possibilities available.

Example 1 — Monitored Move

The name, **Monitored Move** Behavioural Module, describes a set of Behavioural Modules which fall under the movement category. The movement performed can be any one of the VAL-II commands: MOVE and MOVES (straight line or joint-interpolated motion between two locations); APPRO and APPROS (straight line or joint-interpolated motion to a z tool offset distance, approaching a given location); or DEPART and DEPARTS (straight line or joint-interpolated motion to a z tool offset distance, departing from a given location). The sensors monitored during this move, depending on which Behavioural Module is called, can be either a single sensor, or two, four or six sensors.

These Behavioural Modules were designed as low level monitored actions which could be reused in many different situations to build up other levels of Behavioural Modules, increasing in competence up the hierarchy. They can perform either straight-line motion, or joint-interpolated from the current robot position to a position named as a parameter to the

Figure 5–20: Turning regrasp part through 180 degrees (after).

Move Round Assembly

This particular Behavioural Module, **Move Round Assembly**, was introduced to deal with the problem of height clearance when dealing with high assemblies, especially a problem with the introduction of parts from the Soma-5 set. The addition of the force sensor table to build the assembly on also reduced the height clearance in an already shallow workspace.

If the already constructed assembly reaches a certain height, determined by information from the plan in conjunction with the size of the cubes (only known at run-time), the robot will attempt to perform a different **Place** Behavioural Module. A path is determined using information about the already completed assembly and the cube size which will allow the robot to move around the assembly and bring the part in from a direction which will not crash into the parts which have already been inserted. Once the part has been moved around the assembly, it can then be lowered into place as the assembly returns to the Ideal Execution Path.

Figure 5–19: Turning regrasp part through 180 degrees (before).

enable the robot to perform this action. An example of this type of turning is shown in the two photographs in Fig 5–19 and Fig 5–20.

Slip

During any motion with the Soma part held in the Adept gripper fingers, the four sensors between the fingers are monitored to check if the part is still in position. If the sensors detect that the part has slipped, a simple Alternative Execution Path is entered to try to restore the part to the correct position in the gripper.

This strategy currently involves moving the Soma part back to its previous location, either the taught pick up location, or the location on the regrasp pedestal, dropping the part again, then attempting to pick it up by the same method as previously used.

tween the fingers are also constantly monitored to detect the presence/absence of an object.

No motion strategies are used to improve the reliability of the Behavioural Module **Regrasp Put** during this task, even though added variation may occur if the regrasp table is unsteady or the part is uneven. To increase the reliability of the Behavioural Module **Regrasp Get**, part of the same strategy which was used for the **Single Pick** Behavioural Module is incorporated, where the gripper straddles the part until the bottom two lightbeam sensors are obscured (see in Fig 5–14). The two probe sensors are also used as, if they are triggered, the part will have turned sufficiently so that the gripper can no longer straddle the cube. Although, in the current system, no Alternative Execution Path has been devised, this is the type of situation which could be detected and dealt with in that way.

Not only is regrasp an example of an Alternative Execution Path, but the execution of it can take several different paths as is shown in the next section.

Regrasp Twist

If the workcell restrictions are such that the Soma part cannot be placed on the regrasp pedestal with the gripper orientation required (see the problem described in Section 5.1.1 on page 114), the part can be rotated through 180 degrees and put down in this orientation instead. Whether the robot can reach a given point can be worked out by the robot controller at run-time using information from the plan, the taught regrasp location and its own internal sensors. This Alternative Execution Path can be used if at least two approaches to the regrasp table are available, at a 90 degree angle to each other.

Once more, if the gripper cannot reach the Soma part to pick it up in the correct orientation for placing into the assembly, an intermediate rotation of 180 degrees can be used. For this rotation, more information must be supplied to the Prolog plan from the planner to give an alternative grasp location to

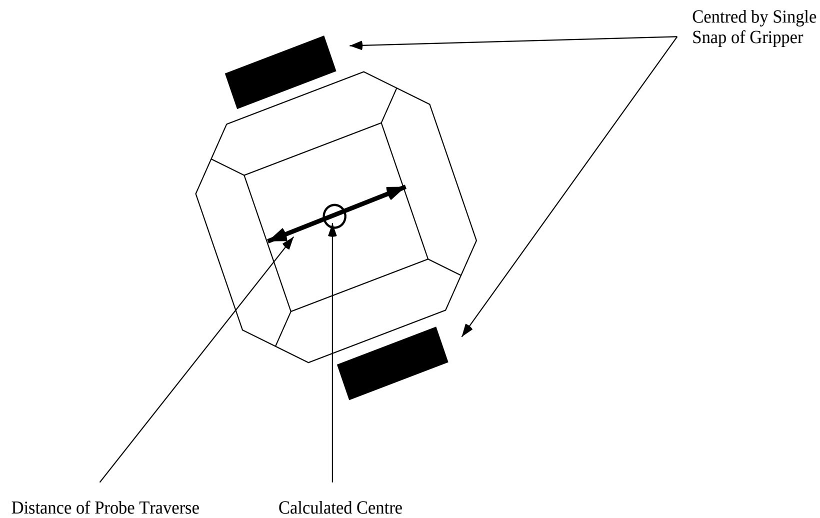


Figure 5–18: Second strategy to centre the Soma part in the gripper.

The regrasp strategy allows a part to be placed on a small pedestal in the workcell and picked up from another orientation, either grasping the same cube, or changing to another specified by the planner. This strategy is described in Section 3.2.4, with photographs showing an example gripper position change in Fig 3–9 and Fig 3–10.

The planner provides details of the orientation of the gripper on placing the part on the table, and offsets which allow the part to be placed in a gravitationally stable position in the centre of the small table. The same is provided for picking the part up again, with details of which cube is to be grasped. Some orientations of the gripper are not possible as the robot only has five degrees of freedom, but these are taken into account when working out the way the part can be grasped.

The part is released on the table then the Adept moves away, re-orients for grasping the part from another direction, and moves back in to grasp it. The probe sensors in the fingers are constantly monitored during all the motions to detect if the probes have bumped into anything and the four lightbeams be-

Figure 5–17: First strategy to centre the cube in gripper.

the above way to find the vertical edge of the cube without bumping into the other cubes at the same height. The strategy then developed used the probe sensor to measure the top surface of the cube, finding the centre by dividing this in two. This strategy is shown in Fig 5–18.

Finally, as in the Behavioural Module **Pick Double Snap**, the Adept is moved down over the Soma part until the bottom two sets of lightbeams are obscured, leaving the top two clear, as is shown in Fig 5–14.

Regrasp

Unlike the part acquisition and part placement which are required for every part used in the assembly, the regrasp is required for just those parts which cannot be placed in the assembly with only rotation around the z axis from the pick up orientation (as in the SOMASS system). This is used as an Alternative Execution Path, identified by knowledge from the planner.

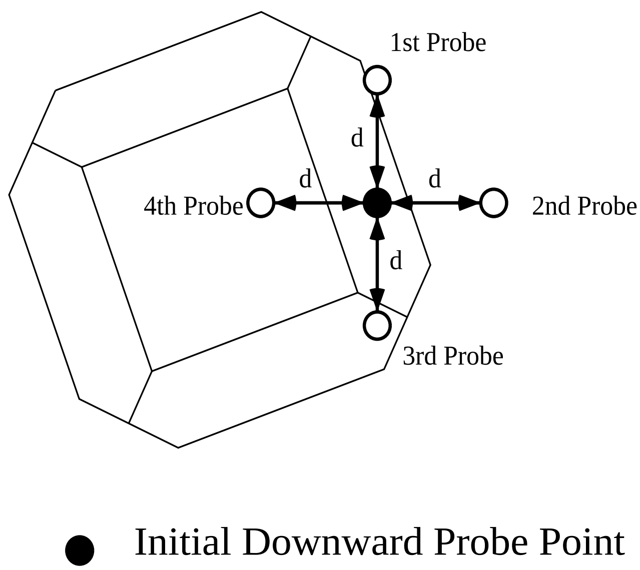


Figure 5–16: Strategy for finding cube top (having found a chamfer).

Single Pick

In some cases, especially when parts from the Soma-5 set are introduced as assembly components, it is not possible to pick them up by a single upstanding cube using the **Pick Double Snap** Behavioural Module. These cases can be identified by looking at the Prolog list representation of the part and checking for any other cubes at the same height in the Soma shape as the one to be gripped. In order to centre the part in the gripper to reduce the locational variation once more, a single snap of the gripper can be performed for one direction, but the other requires a different strategy because of the extra cubes in the way.

The original strategy devised for this was to use the four sensors located between the fingers to find the vertical edge of the cube, as can be seen in the photograph in Fig 5–17. Once the edge was found, the gripper could calculate the centre of the Soma cube. This proved an impossible strategy when dealing with some of the Soma-5 set where the part had cubes located on a diagonal to the one to be grasped. This meant that the gripper could not be used in

Regrasp Reorient the Soma part for placing in the assembly.

Regrasp Twist Turn the Soma part on the regrasp pedestal if the Adept cannot reach the cube to be gripped in the correct gripper orientation.

Slip If the Soma part slips in the gripper while being carried, try to regrasp the part.

Move Round Assembly If the assembly becomes too high to allow clearance of parts lifted over it, the parts can be placed in the assembly after moving round the already completed assembly.

The extent of the Alternative Execution Paths varies from small deviations to check a sensor reading, to changes involving the regrasp of a part. It is only some of the major alternatives which are detailed here. More detailed information of the Behavioural Modules will be described in Section 5.5, as the current section deals mainly with the strategies involved.

Find Centre

If the Behavioural Module, **Find Height**, fails to land on the top of the Soma cube to be gripped during the strategy described in Section 5.1.2, landing instead on the chamfer of that cube (determined by comparison with the nominal height of the cube), a search can be made for a point located on the top of the cube as it is known to be nearby.

The search strategy which is employed in the Behavioural Module **Find Centre** simply probes downward a short distance from the chamfer point along each of the four axes directions ($\pm$ x and $\pm$ y) until a point corresponding to the nominal cube height is found, as seen in Fig 5–16. This point can then be used as the central point for the **Location** Behavioural Module described in Section 5.1.2. Control then returns to the original Ideal Execution Path and the system proceeds as before.

Figure 5–15: Using the Adept wrist to push the Soma part into place.

tion Paths which can attempt to bring the system back from a certain error situation.

The Alternative Execution Paths described below have been developed to cope with specific failures in the Ideal Execution Path. Sensory data and planner information in conjunction with knowledge about the physical workcell can be used to determine when these Behavioural Modules are appropriate, and the particular hierarchical structure of the linking of the Behavioural Modules means that these can be easily integrated into the system.

The following sections detail some of the major Alternative Execution Paths used in the system.

Find Centre Search for a point on the top of the Soma cube if a chamfer has been detected on first downward probe.

Single Pick Centre the Soma part in the gripper if **Pick Double Snap** Behavioural Module cannot be used.

workcell. The planner provides details of offsets from this position depending on the placing of the individual parts as described in Chapter 3, page 73. The actual distances are calculated at execution time and are dependent on the size of the cube.

The part is always inserted downward into the assembly. When the force sensor table is used as an assembly platform, the force/torque readings can be examined to determine when the part has come into contact with the table or the assembly, and the robot can be halted. At present, only the z reading is examined to determine this contact, with a certain threshold defined as contact having been made. The force sensor could be used in many other ways to obtain more information about how the contact is made. Once contact has been achieved, the gripper is opened, closed then opened once more to make sure the part is correctly in place. The robot then retires to a safe height once more. The sensors are constantly monitored for difference from expected values and unexpected transitions.

As chamfered cubes are used for the assembly, it is no longer necessary to leave pads in the assembly to cope with positional variation as the chamfers themselves will prevent parts catching on the tops of others (see Chapter 3, page 67). It is, however, still necessary, with some assemblies, to leave a gap for finger clearance when inserting the last part. This part can then be pushed into place using the back of the robot wrist. The photograph in Fig 5–15 shows this strategy. This is only required for shapes like the large 3 x 3 x 3 cube where no space is available for the grippers to insert the parts otherwise.

5.1.3 System Overview — Alternative Execution Paths

The Behavioural Modules which have been described fall within the Ideal Execution Path of the system for the application of putting together Soma assemblies. These Behavioural Modules are quite reliable (see Section 6 for experimental runs of these Behavioural Modules), but still fail to perform their task on occasions. On these occasions, it is useful to have Alternative Execu-

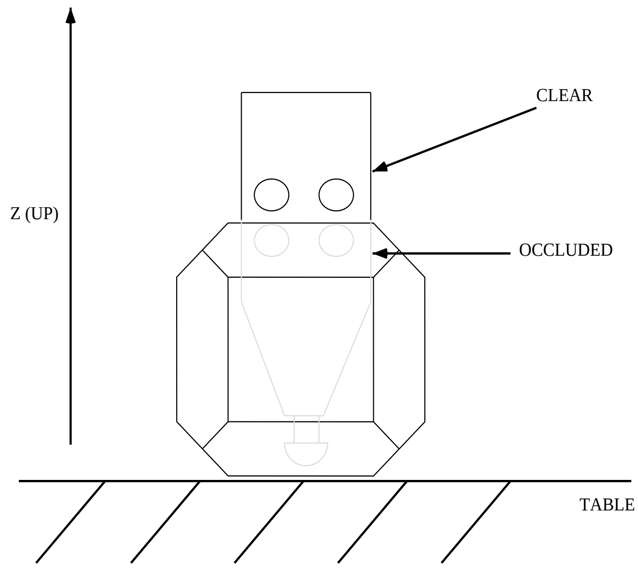


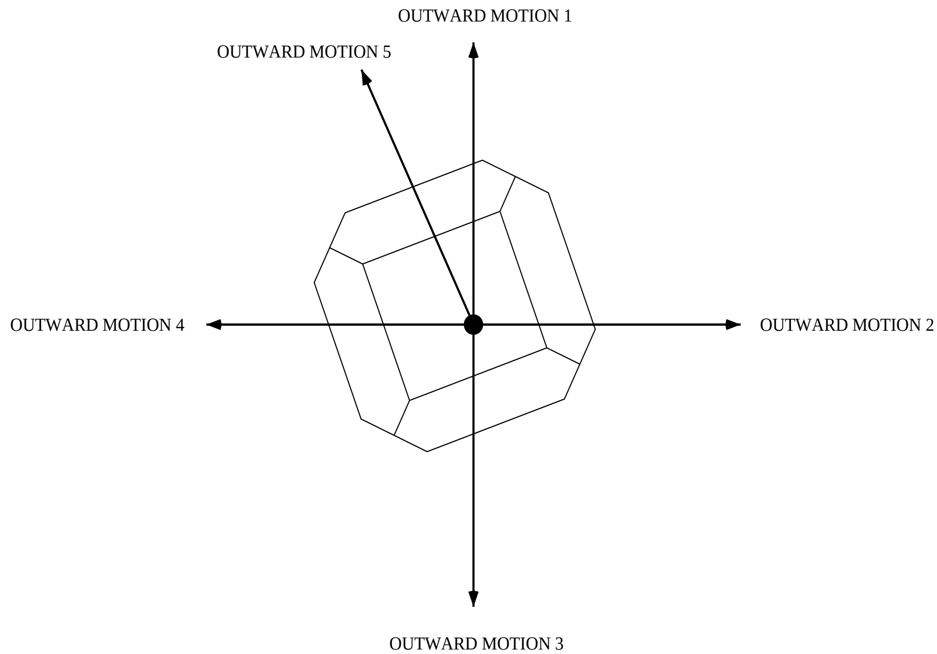
Figure 5–14: Final position of the lightbeam sensors on part pick-up.

with its fingers, without colliding with the top of the part. The results from the experimental system runs in Chapter 6 give more details of the variation that can be dealt with.

The gripper moves to a safe height over the calculated centre point of the cube, then vertically descends over the part until the bottom two sensor light-beams between the fingers are broken by the cube. This is performed in the Behavioural Module **Pick Double Snap** which is detailed in Section 5.2. The cube is centred in the gripper with the double snaps as previously shown in Fig 3–16. The part is then gripped with the lower two lightbeam sensors in the fingers obscured by the gripped cube, while the upper two are clear of the top of the part (as shown in Fig 5–14), and moved to a safe height in the workcell.

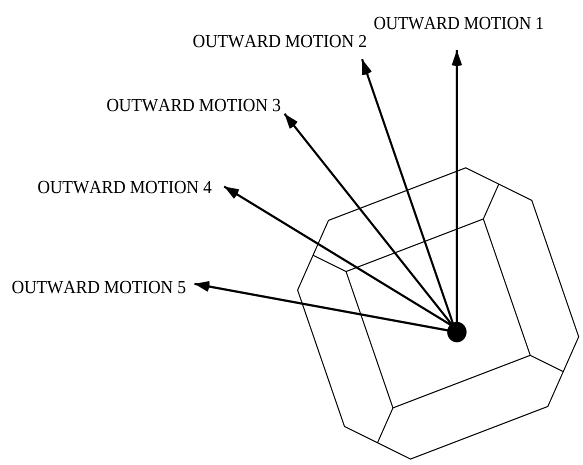
Part Placement

The part placement Behavioural Module, **Place**, requires that a location is taught to the system informing it where the assembly should be built in the



● Initial Downward Probe Point

Figure 5–12: Finding orientation of a Soma part — Strategy 1.



● Initial Downward Probe Point

Figure 5–13: Finding orientation of a Soma part — Strategy 2.

Figure 5–11: Gripper after downward probe to locate cube top.

the point already found on the top of the cube, and stopping the robot by triggering the sensor when the probe was released when it fell off the edge of the cube (consistently halfway down the chamfered slope). These five points were enough to determine an approximation of the orientation of the cube except in the case where the angle was near zero. This was due to the method of calculation (sine of a small angle), and it was decided that a complete new strategy could be devised which would prove more useful than finding an alternative strategy when the angle was near this problem orientation.

Using the same method of finding the edges of the cube, the second orientation strategy found a series of points along the edge until the orientation changed to show a corner had been turned. See Fig 5–13 for an example of such a strategy. When a corner was located, the centre of the cube could be estimated sufficiently accurately from this and the knowledge of the nominal size of a cube.

The strategy implemented is not a perfect way to find the orientation of the cube, but performs well enough to enable the robot to straddle the cube

cube by using suitable motion strategies and the binary sensors in the fingers.

Part Placement: The Soma part can be placed in the final assembly in a location calculated from offsets from a nominal taught location. Variation can be dealt with by the use of the chamfers on the cubes, by the force/torque sensor table, and using the sensors between the fingers.

The following two sections detail the strategies used to deal with variation and uncertainty in this assembly system, but look only at the Ideal Execution Path through the system. Section 5.1.3, examines the strategies employed when these Ideal Execution Paths fail, and an Alternative Execution Path may be taken instead.

Part Acquisition

The part acquisition Behavioural Module, **Pick-up**, requires taught nominal locations for each part, and the nominal size of the cubes in the part. The implemented strategy makes use of guarded motions performed using the two probes in the Adept fingers, and the four sensors between the fingers.

Initially, the top of the cube to be gripped is located by performing a guarded motion downwards towards the cube with one of the probe sensors using a Behavioural Module called **Find Height** (detailed in Section 5.2). The photograph in Fig 5–11 shows an example of the position of the gripper after one such motion. The case where the probe lands on a chamfer is dealt with in Section 5.1.3. Once the top of the cube has been located, the robot performs a motion strategy to determine the orientation of the cube using the Behavioural Module **Location**, which is again detailed in Section 5.2. The first strategy designed to cope with this found the horizontal edge of the cube along each of the $\pm$ x axes and $\pm$ y robot axes, and an additional fifth point to distinguish between ambiguous cases. Fig 5–12 shows an example of this strategy. The points were found by moving outward along the robot axes from

Figure 5–10: Force/Torque Sensor Sandwich.

5.1.2 System Overview — Ideal Execution Path

This section outlines the strategies used on the Ideal Execution Path of the experimental system. From this point onward, the experimental system referred to will be the one described in Section 5.1.1, using the set of gripper fingers described in Experimental System 2 and the control from Experimental System 3.

This experimental system was designed to perform the assembly of chamfered Soma-4 and Soma-5 parts, constructing pre-selected assembly models. The task decomposition falls into two stages for each part in turn: acquiring the part, and then placing it in the assembly. Each of these is performed by a Behavioural Module which in turn decomposes the task into smaller sections. Both of these Behavioural Modules fall on the Ideal Execution Path.

Part Acquisition: The Soma part can be acquired from a nominal taught location, dealing with variation in orientation of approximately $\pm$ 45 degrees and translational variation equivalent to the size of the top of the

Figure 5–8: Adept robot workcell limitations — Gripper rotation 2.

Figure 5–9: Adept robot workcell limitations — Gripper rotation 3.

Figure 5–7: Adept robot workcell limitations — Gripper rotation 1.

positions as in Fig 5–9, but cannot reach those in Fig 5–7. This can lead to problems when parts are placed in position with the gripper in one orientation, and picked up with the gripper in another. This problem is compounded by the addition of the force sensor attached to the wrist. The z height to which the Adept can move with the gripper oriented as in Fig 5–9 is also severely limited.

The second problem involved the noise of the robot vibration in the sensor readings when it was mounted on the arm. To reduce these two problems, the force/torque sensor was sandwiched between two metal plates and the assembly was built on top of the plates, as can be seen in the photograph in Fig 5–10. This meant that the force/torque sensor could not be used in the acquisition of the parts, but the binary sensor coverage of this was adequate for the task.

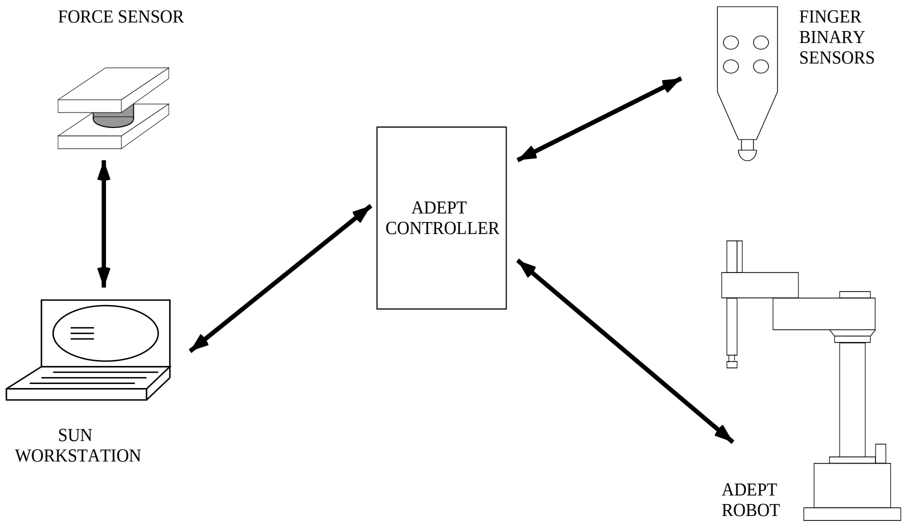


Figure 5–6: Communication within the system.

helped with the integration of other sensing systems which were already connected into the backplane of the SUN, such as the force/torque sensor. Work on a vision system [Chongstitvatana & Conkie, 1990], using Datacube software, could also have been incorporated if required, but this type of sensing was not used in the experimental system.

A brief mention should be made here concerning the use of the force/torque sensor. Connected directly into the backplane of the SUN workstation, the same communications program used for the Adept controller could be modified to read the data from the force sensor. Although the force/torque sensor was originally designed to be attached to the wrist of the Adept, this caused two separate problems.

The first of these was the length this added to the wrist in the limited workspace of the Adept robot. (See the photographs in Fig 5–7, Fig 5–8 and Fig 5–9.) In Fig 5–7, with the Adept arm fully extended, the gripper can reach a position much further away than is possible after a simple rotation of the gripper though 180 degrees, which is shown in Fig 5–9. If the gripper is positioned vertically downwards, as shown in Fig 5–8, it can reach the same

that the output from the planner, the Prolog list, could be interpreted as the system ran. This gave the advantage of quick testing of modifications to the interface between the planner and the Behavioural Modules, easy integration of other sensing systems which had been developed, such as force sensing or vision, and editing the Behavioural Modules became much easier, as the editing facilities on the robot controller had proved unwieldy when dealing with large systems as compared to the advanced editing facilities on the SUN workstation.

A C program had been developed by A. Conkie (EDAI) to allow simple communication between the SUN workstation and the Adept controller through an RS232 serial line. This allowed a single command to be sent to the controller, control returning to the SUN when the robot signaled completion of the command. This program had to be modified to allow return of control to the SUN workstation during the execution of the command, such as a MOVE instruction, to allow other sensors which were connected directly to the SUN workstation to be interrogated.

Fig 5–6 shows the new system set-up. As the VAL-II code could be run one line at a time from Prolog, this brought about a more flexible control environment, and the structure of the system could be changed as required for experimentation. Control decisions could be made at any point in the program, based on information from the planner, or on information from the sensors. The robot controller could still deal with the kinematics and mathematics of the robot, and the Prolog could deal with the interface with the planner and the control structure. The binary sensors could still be used directly through the controller, allowing fast reaction time, and other sensors, such as a force sensor, could be incorporated at any level of the new structure.

Many Behavioural Modules had already been written in VAL-II code and stored on the Adept controller. These were not updated to run directly from SICStus Prolog (although they were now accessed as subroutines by a direct call from SICStus Prolog). All new development of Behavioural Modules was subsequently handled from the SUN. All VAL-II commands, even the calibration of the robot, could be dealt with from this new integrated system. This

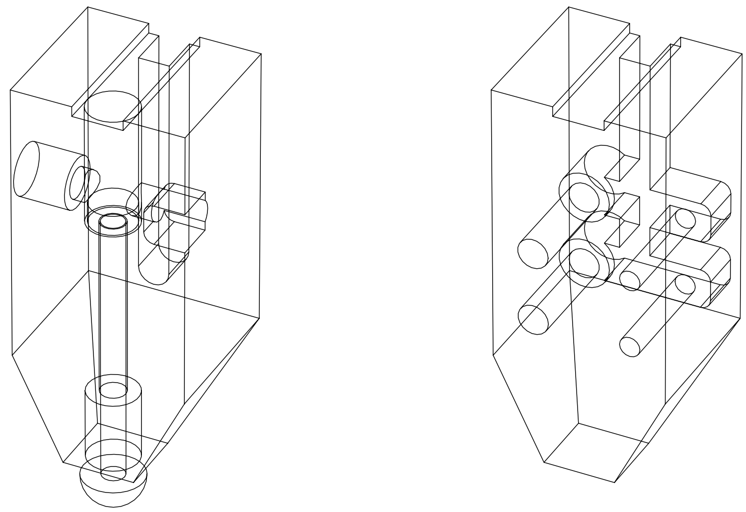


Figure 5–5: CAD model of the finger design.

The two sets of fingers which were designed were used within Behavioural Modules which were stored and run directly from the Adept controller, as were the Behavioural Modules designed for the SOMASS system.

In order to provide an easier working environment where the plan could be run directly from a SUN workstation instead of sent down manually to the controller and stored there, the system was changed once more. The following section details these changes.

Experimental System 3

To aid the program development and provide a more flexible working environment, it was decided that the complete system should be run from the SUN workstation. The SOMASS system, as has been described in Section 3.2.2, provides Prolog plans from a SUN workstation which are sent down an RS232 serial line to the Adept controller. These plans contain calls to Behavioural Modules which are stored within the controller. Much time was spent sending these VAL-II plans to the controller, and a slight modification in plan structure or level of interface required a new VAL-II file to be produced and transported. With the whole system run from the SUN workstation, individual VAL-II commands could be sent down from Prolog (SICStus Prolog [Sic1988] was now being used for the planning system) to the controller at run-time, which meant

Figure 5–3: Second set of gripper fingers (view A).

Figure 5–4: Second set of gripper fingers (view B).

Experimental System 2

To maximise the information which can be gained from binary sensors, and having observed from previous experience with the first set of sensors what information can be useful in the manipulation and assembly of Soma shapes, the new set of fingers, incorporating six binary sensors was designed.

Infra-red lightbeams replaced the microswitch, which was too bulky, inaccurate, and easily damaged. The photographs in Fig 5–3 and Fig 5–4 show the sensor arrangement on the fingers. Each finger incorporates a light beam across the top of the gripper which can be broken by depressing the probe at the end of the finger. Fig 5–5 shows a CAD model of the probe itself and the finger design which shows where the sensors actually lie. Each finger also holds two lightbeam emitters and two detectors which line up with the detectors and emitters respectively in the opposite finger. Again, the finger design to hold these sensors is shown in Fig 5–5. The two sensor set-ups which incorporate the probe will be referred to as the probes to distinguish them from the four lightbeam sensors between the fingers.

The binary sensors were located in the fingers with the following design features and functionality in mind:

- Ability to detect a point of contact of the gripper fingers with any object in the z-downwards direction, stopping the robot if necessary.
- Find a vertical or horizontal edge of the object, again stopping the robot if necessary.
- Detect presence/absence of object in gripper.
- Detect slippage of object in gripper.
- The use of chamfered fingers and rounded probes so no snagging can take place.

the robot immediately (or as fast as inertia will allow), and allows diversion to a subroutine if required.

This kind of sensor triggering which can halt the robot allows guarded motions to be performed. The stopping conditions can be any sensor transition, or combination of sensor transitions (both OR and AND combinations), in this case, either opening or closing the microswitch, or breaking or completing the lightbeam. Once the robot has been stopped, its location in the workcell can be returned to the controller and this information used by the system. With the lightbeam sensor between the fingers, the gripper can be constantly checked to see if it is holding a part.

Although useful in the first experiments with the system, several problems came to light concerning the use of these fingers and sensors. The microswitch proved to be a bulky addition to the gripper, often catching on the parts during the assembly and causing damage to the parts and the microswitch itself. This was particularly a problem when the parts were being inserted into the assembly. The use of the lever on the microswitch (again, see the photograph in Fig 5–2 for this detail) gave rise to problems in accuracy when using the microswitch for guarded motions, as the lever could trigger the microswitch from any point along its length.

Along with the above problems, it soon became clear that much more useful information could be gained by modifying the design of the fingers. The microswitch was replaced by a simple lightbeam arrangement to protect the bottom of the fingers, and the finger bottoms were chamfered to prevent snagging. More lightbeams were incorporated between the gripper fingers. Overall, the fingers were less bulky, had more sensory capability, and were better protected from damage. The next section looks at these new fingers, and the information which could now be derived from them.

Figure 5–2: The original gripper fingers.

gripper fingers. When located in the fingers in this way, both sensors can be used to constantly monitor aspects of the robot gripper in the workcell, especially when it is moving. The motion of the robot can also be used to acquire information by using dynamic sensing. Some sensors are usefully placed in the workcell itself, especially those sensitive to robot vibration, such as a force sensor. Work done in Aberystwyth [Sharp, 1990] [Loughlin, 1992] has produced some useful touch sensors which can be placed around the workcell.

The VAL-II REACTI command, an extension of the basic REACT command which allows the robot arm to be halted when sensors are correctly triggered, can be used to initiate the constant monitoring of a binary signal. If a signal is monitored to detect a transition from 'off' to 'on', and the signal is already 'on', no triggering occurs until the signal has gone to 'off' and then 'on' again. The signal must remain stable for at least 18 milliseconds to assure detection of the transition. The monitoring continues until either the correct transition occurs on a particular sensor, or until an IGNORE command is issued for that sensor. When the sensor is triggered, the REACTI command stops

description for the new Soma-5 set, and descriptions of the expanded range of final assembly shapes. In dealing with final assemblies which used many more parts, the time factor for finding a solution increased many times due to the combinatorial expansion. For some of the larger assemblies (composed from more than the original seven Soma-4 shapes), no solutions were found after several days.

The aim of the newly expanded system was to introduce sensing into Behavioural Modules and to examine their subsequent linking and composition. As the ideas developed, several major changes took place in the system, and these are detailed below. The changes reflected both the need for sensor information appropriate to the task in hand, and the need for a more flexible programming environment. It became quickly apparent that the software, hardware, and the task were closely inter-related.

Experimental System 1

The first experimental system which was developed incorporated simple binary sensing into Behavioural Modules. The Adept controller is configured to handle eight external binary input/output signals through a serial line. Each signal can be set to either 1 or 0, signifying 'on' or 'off'. These designations of 'on' or 'off' are dependent on the electronic set-up of the particular sensor. The binary sensors can be monitored directly by several VAL-II commands, such as REACT, SIG, and SIGNAL. REACT commands initiate the monitoring of the binary signals, SIG returns their state and SIGNAL can be used to trigger the sensors. See the Adept User Manuals for specific details of these commands as used with the Adept robot [Ade1985a] [Ade1985b].

The first sensor development integrated into the system involved two binary sensors, one a microswitch, the other an infra-red lightbeam emitter/detector set. The photograph in Fig 5–2 shows the placement of these sensors on the gripper. The microswitch is located at the tip of one finger, and the lightbeam emitter/detector set are placed such that the lightbeam crosses between the

assembly. If the assembly has reached a certain height, instead of moving directly over the assembly and perhaps colliding, extra work can be done to find a safe route into the assembly for the part, avoiding the already completed sections. This comes in useful when the Soma-5 set is introduced, or when tall assemblies are attempted. For the insertion of the parts, a force/torque sensor is used to detect when the part reaches the table.

Most important for this particular work is the increased reliability which can be gained by the use of the Ideal Execution Path and Alternative Execution Paths. Information to determine change of control direction to the Alternative Execution Paths from the Ideal Execution Path is available from Exit States and local relevant information.

The following sections examine the implementation stages of the experimental system, and the strategies used in the Behavioural Modules in the final system on both the Ideal Execution Path and the Alternative Execution Paths.

5.1.1 Implementation Stages

This section looks at the system development and introduces in more detail the sensory system and the programming environment. As the experimental system developed, several important modifications were made to its hardware and software. The sections below detail the three major changes which were required to improve the system and provide a better test-bed for the work. It is also important to note how these hardware and software changes were affected by the nature of the task itself.

The workcell's main component, consistently used through all the experiments, is the 5-degree of freedom Adept robot (shown in Fig 3–7). Chamfered Soma shapes from the Soma-4 and Soma-5 sets are used as the assembly components. As with the SOMASS system, the final assemblies are shapes constructed from the individual Soma parts. Some of these can be seen in Fig 3–4. The SOMASS planner was retained in the same form as before (see Section 3.2.3), with the only addition being the input parameters of the part

Figure 5–1: A Chamfered Soma-4 Shape — Fork3.

been relaxed, such as the strategy of leaving pads in the final assembly, as the system uses chamfered Soma parts (see Fig 5–1) which can adequately cope with small translational variations.

The experimental system, as with the SOMASS system, uses some motion strategies to reduce the variation present, but it also incorporates some sensing strategies to help increase the reliability. The sweeping motions to move the parts into a predefined position have been replaced by simple strategies using binary sensors to acquire the part. Where possible, the double snap of the gripper is used to centre the part in the gripper (see Fig 3–16), but when this is not possible, the part is centred in the gripper by using a strategy involving the simple sensors (see Section 5.1.3 on page 125). Whenever the part is carried in the gripper, binary sensors monitor the contents of the gripper. If regrasp is necessary, this is also closely monitored, allowing for more variation than in the sensorless SOMASS system. When inserting the parts into the assembly, the parts which have already been assembled are taken into account to ensure that the part carried in the gripper will not collide with the already completed

The SOMASS system, described in Chapter 3, was selected as a foundation for the development of Behavioural Modules using sensors. The advantages of the Soma shapes as examples of assembly objects are examined in Section 3.2.1. The existing planner for the SOMASS system was retained, with the focus of the work concentrating on the development of appropriate sensory Behavioural Modules and their linking. The planner output of a Prolog list remained unchanged, see Appendix A, but was post-processed into a different form more suitable to exploit the new structure and sensors available.

The first section of this chapter examines the implementation details and the structure of the experimental system, giving a complete outline of the strategies and Behavioural Modules used. The second section looks in more detail at some of the implemented Behavioural Modules. The third, fourth, and fifth sections examine the Exit States of some of the Behavioural Modules, the Ideal Execution Path, and some Alternative Execution Paths of the system respectively. The final section provides a summary of the work.

5.1 System Outline

The system for the thesis work, based on the SOMASS system, performs the task of assembling Soma structures. The planner for the SOMASS system is retained, but the structure and content of the Behavioural Modules are changed. Simple binary sensing is integrated to allow decision making capabilities dependent on sensory information.

As with the original SOMASS system, the planner implicitly uses some rules to facilitate assembly and assure that certain motions which are deemed unreliable are not used. The rules which are still enforced in the experimental assembly system include downward acquisition and insertion of the parts. Parts must still be gravitationally supported when inserted into the assembly, with each cube in the Soma part supported by the floor or another cube. Again, this forbids the assembly of any shape with an overhang. Other rules have

Chapter 5

The Experimental System

This chapter examines the experimental system which was designed and implemented to test the ideas related to Exit States, Ideal Execution Paths, and Alternative Execution Paths (see Chapter 4). The Behavioural Modules developed for this system provided a basis for examining the details which would be needed to link them successfully to build a reliable and robust assembly system.

As it is too soon in the development of the field for a coherent theory of robotic assembly to have been formulated to provide a framework in which the Behavioural Modules could be tested, the experiments invented for the system were designed to test its ability to successfully complete a task in the presence of variation using the Behavioural Modules on the Ideal Execution Path. If it could not do this, alternative Behavioural Modules were provided for parts of the system, and if these did not allow the task to be successfully completed, the system was allowed to fail, providing details about this failure.

The experiments detailed in Chapter 6 show two sets of runs of the experimental system, one using only the part of the system which acquires Soma parts, and the second set showing the running of the whole system. The number of times the Ideal Execution Path and the Alternative Execution Path were used in both trials are noted, along with the failures and the reason for the failures.

information decree that the system is no longer in a state to follow the Ideal Execution Path, Alternative Execution Paths may be available to bring the system back to the Ideal Execution Path. Alternative Execution Paths may be used to deal with more, or different, variations in the workcell, or may be used to recover from a potential error situation.

The key to the diagrams which will be used throughout the thesis to illustrate the Behavioural Modules has also been presented.

composed of other Behavioural Modules. If the Behavioural Module, TASK-A, is composed of Behavioural Modules, TASK-B and TASK-C, the failure may be due to either one. In this case, as in Fig 4–5, there is no alternative available at the TASK-B and TASK-C level, the problem may be left to be dealt with at the level of TASK-A and TASK-A2. This time, TASK-A has the choice of three values to return, S, F1, or F2. The difference between F1 and F2 may be relevant at a higher level to determine where the Behavioural Module went wrong. In this diagram, it has been decided that if the returned value is F1, the alternative Behavioural Module, TASK-A2 may be attempted. These values may be carried up as far as required as Exit States from each level of Behavioural Module.

4.6 Reliable Behaviour-based Robotic Assembly

This chapter has introduced the terms and ideas which are used in this thesis to investigate building a reliable robotic assembly system. These include Exit States, the Ideal Execution Path, and Alternative Execution Paths.

A Behaviour-based robotic assembly system can be split into two distinct sections, comprising a planning system and an execution system. The planning system provides a high level description of the part motions, and these are realised on-line by a competent execution system composed of Behavioural Modules. These Behavioural Modules are modular units which combine software and hardware to perform a task in the real world and can deal with a certain amount of variation and uncertainty in the workcell. Exit States, which inform the system what has occurred within a Behavioural Module, can be used along with local information to identify the Ideal Execution Path and appropriate Alternative Execution Paths. The Ideal Execution Path is a control path through the hierarchy of Behavioural Modules which describes the best route through these as decided by the designer of the system. If sensors or local

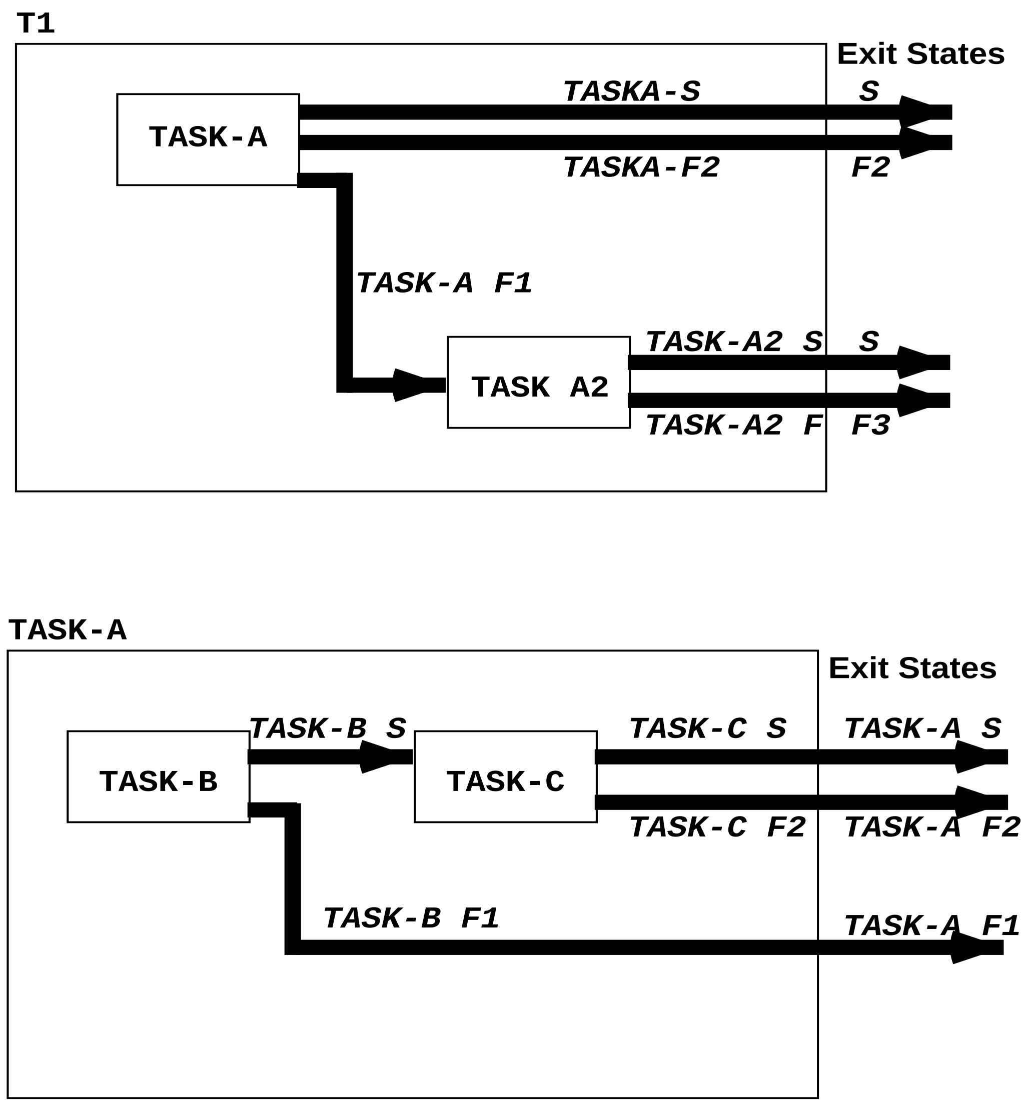


Figure 4–5: Multiple level Alternative Execution Path.

ferent sensors, or may simply try the Behavioural Module, TASK-A once more. The Behavioural Module, TASK-A2, will return the value S, or F. This time, if F is returned, the system designer has decided that there is no alternative which can be tried. In this case, the Behavioural Module, T1 will return F. Thus, this example demonstrates how an alternative can be tried at the same level as a failure occurs.

The next example in Fig 4–5 looks at a deviation from the Ideal Execution Path which cannot be dealt with at the same level of Behavioural Module. Again, using T1 as a basic task, this looks at a slightly more complex situation. TASK-A may consist of either local robot command coding, or may in turn be

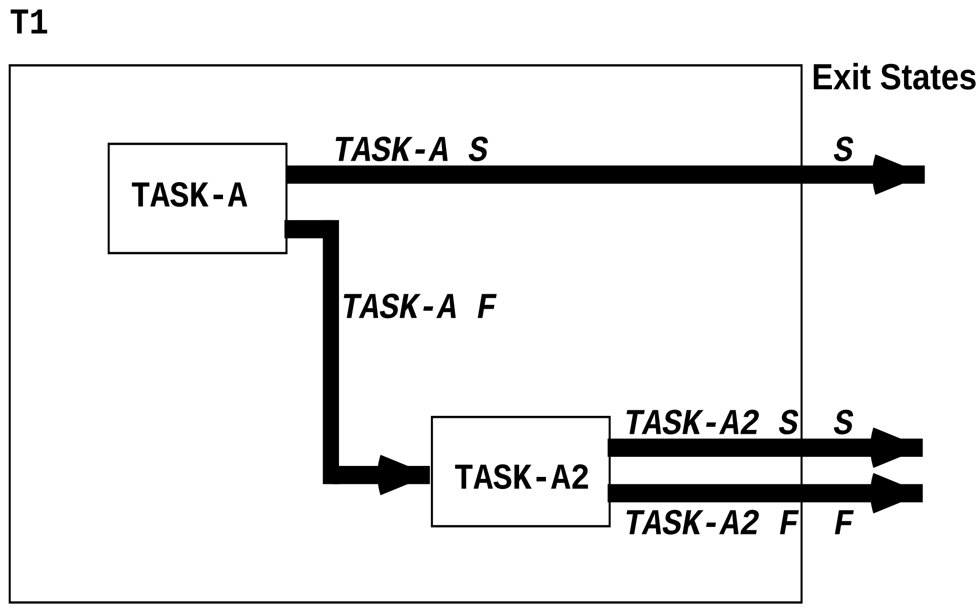


Figure 4–4: Single level Alternative Execution Path.

the problem is dealt with, the greater the problem there is with the variation and uncertainty in the workcell. As with the hierarchical structure devised by Albus, the problem of variation can be dealt with at any level, although with this system, the decomposition is simpler and the sensory structure different. The following example shows how failure can be dealt with at different levels of the system.

In Fig 4–4, the Behavioural Module, named T1, has been designed to complete a task, TASK-A. The Behavioural Module, named TASK-A, is executed and can return one of two values as its Exit State, in this case, either S or F. As this is a simple task, these will currently be referred to as Success or Failure to make explanation of the example easier.

If TASK-A returns Success, then the Behavioural Module, T1, will return Success. On the other hand, if the result of performing TASK-A is Fail, then an alternative Behavioural Module, TASK-A2 can be tried. TASK-A2 is designed to bring the assembly to the same state as would occur at the successful completion of TASK-A. The Behavioural Module TASK-A2 may involve performing a different strategy with the same sensors as TASK-A, it may use completely dif-

a task. Extensive repeated deviation to Alternative Execution Paths shows that the Behavioural Modules on the Ideal Execution Path are not now the best suited for the task, whatever the criteria for their original choice. Either the Behavioural Modules on the Ideal Execution Path have proved inadequate to deal with the task, or the circumstances in the workcell have altered and may need attention. In a system with no sensors and no information available concerning possible Alternative Execution Paths, control will automatically follow an Ideal Execution Path. Even if the variation proves too much for the task to be successfully completed, the robot will still go through the motions of the assembly with no parts in its gripper.

The linking of the planner with the Behavioural Modules provides an interface which is useful in determining the reasons for the failure of an assembly system. The failure could be due to either the use of inappropriate Behavioural Modules, or due to the level of the interface with the planning system providing inappropriate information for the use of these Behavioural Modules. In other words, the question becomes one of discovering whether the Behavioural Modules are competent enough to deal with the variation in the environment, or whether the planner is perhaps producing the wrong level of detail for the Behavioural Modules which have been selected. If a large number of Alternative Execution Paths are used regularly, then perhaps the choice of Behavioural Modules on the Ideal Execution Path is inappropriate for the task.

Failure at Different Levels

The use of Alternative Execution Paths, deviated to by information from Exit States and local information, allows the system to deal with the potential failure of the system at any point in the hierarchy. The problem can be dealt with at a low level within the same Behavioural Module, or may be tackled at a higher level where another alternative can be used. This means that there is no ultimate failure of the system until the top of the hierarchy is reached and nothing more can be done. This leaves plenty of scope for recovery at any level when relevant. It is usually the case that the further up the hierarchy

4.5 Usefulness of the system

This section looks at some of the other advantages which can be found when using the above ideas within the framework of a Behaviour-based robotic assembly system, apart from that of providing a more reliable system by using appropriate alternative actions.

Decomposition of the assembly system

In the SOMASS system described in Chapter 3, and the experimental system developed for this thesis (described in Chapter 5), the planner plans for assembly in terms of part motions, rather than in terms of the robot motions required to achieve the part motions. With this comes the advantages of Object Level programming where it is seen as easier for the programmer to think in terms of part motions (see Chapter 2, page 20). The planner, although enforcing the constraints previously discussed, does not know anything about the dimensions of the physical assembly parts or workcell, or the availability of sensors, and performs no reasoning about the variation or uncertainty in the physical workcell. The execution system, composed of Behavioural Modules, interfaces to the planner only through a series of part motions provided by a Prolog list (see Appendix A), and this allows the execution system the freedom to interpret these without resorting to replanning if a particular robot motion is not appropriate. The Behavioural Modules can be easily changed and inserted into the execution system to produce a more competent system using different sensors and different strategies where appropriate. The execution system can be used to handle possible error situations without re-planning.

Suitability of Behavioural Modules

The extent to which the system follows the Ideal Execution Path can be taken as a measure of the suitability of the Behavioural Modules chosen to perform

Having set out the ideas concerning the Ideal Execution Path and the Alternative Execution Paths, the following example shows how these can be used in practice. Supposing the existence of a Behavioural Module whose task is to find an object (referred to as **Find Object** Behavioural Module). If the object is usually in a certain location and orientation, then a simple touch sensor may be used to find it with this Behavioural Module being on the Ideal Execution Path. If this fails due to the actual workcell configuration being outwith the variation it can cope with, an alternative type of sensing, such as vision, may be used, providing an Alternative Execution Path. The competences of the two Behavioural Modules, one using touch and the other using vision, will be different, with vision perhaps able to sense quickly over a larger area, but the touch may be more accurate. So, to the system designer, it may be obvious to try the touch first, and if that fails, try the second. Thus, the **Find Object** Behavioural Module may still succeed, but by a different method from the Ideal Execution Path.

Some Behavioural Modules are only used within an Alternative Execution Path, and thus will never be included in a system's Ideal Execution Path. Any lower level Behavioural Modules within **Behavioural Module 3** in Fig 4–2 will not be on the Ideal Execution Path for the example system. When a Behavioural Module is called in different circumstances in different parts of the system, different routes through it may be included on the overall Ideal Execution Path. One call of **Demonstration** (Fig 4–2) may determine that the Ideal Execution Path for the system follows through **Behavioural Module 1**, with Exit State BM-1A, and through **Behavioural Module 2**, with Exit State BM-2A, whereas another call of the Behavioural Module, still on the Ideal Execution Path, may follow through **Behavioural Module 1**, with Exit State BM-1B, and on through **Behavioural Module 3** with Exit State BM-3B.

the correct feature is identified, but occasionally the Alternative Execution Path containing the search strategy may have to be used.

There follows a list of circumstances where an Alternative Execution Path may prove useful.

1. A Behavioural Module on the Ideal Execution Path may not have the competence to deal with the variation actually present in the real workcell. For example, an object may not be found within the range of the visual field of a vision system being used by a Behavioural Module. Alternative strategies to deal with this problem may range from moving the camera to using other types of sensor, depending on the capabilities of the system.

2. A Behavioural Module on the Ideal Execution Path, on consulting information gained from sensors within that Behavioural Module or from local information, may decide that it is no longer in a situation where it can best deal with the task. For example, a position calculated in the Behavioural Module may be outwith the reach of the robot. This cannot always be known before entering the Behavioural Module and actually calculating the position. In this case, the robot may, for example, be able to approach the position by using a different gripper orientation.

3. If the information received from a Behavioural Module which has successfully completed its task is determined to be invalid for the current application of the Behavioural Module, an Alternative Execution Path may be followed. For example, a value corresponding to a height is returned, thus the Behavioural Module whose task it was to find a height has been successful, but this height does not lie close enough to an expected estimated height. Depending on the context and capabilities of the Behavioural Module, the source of the height which has been found may be easily identified and an alternative strategy used to bypass this obstacle.

The deviation to an Alternative Execution Path is determined by the Exit State of the Behavioural Modules or from local information available to the system. When an Exit State and qualifying parameters are returned from a Behavioural Module, this can be examined to determine whether the task was successfully completed in the context of the calling or whether it failed. The Exit State can show whether the Behavioural Module could not attempt the task due to the conditions in the physical workcell being outwith its competence, or whether it partially completed the task, but then failed to complete it. This is useful information which can be used to determine whether there are other Behavioural Modules which could cope with the unusual situation and bring the system back to the Ideal Execution Path.

Alternative Execution Paths can also be entered into by information from the planner which can be instantiated at run-time with information from the real world. The planning system does not know anything about the dimensions of the parts and the workcell, but the on-line system can determine if alternative Behavioural Modules should be used due to, for example, the height of an assembly becoming too great for the normal method of inserting a part. In Fig 4–2, an Alternative Execution Path is available through **Behavioral Module 3** if the Exit State from **Behavioral Module 1** is BM-1B.

Alternative Execution Paths may consist of Behavioural Modules, or may merely perform local robot commands, depending on the extent of the deviation from the Ideal Execution Path. The Alternative Execution Path may simply retry the original Behavioural Module which returned a failed Exit State if it is possible that it may succeed in correctly performing the task if repeated. The Alternative Execution Path may instead use different sensor types if available, or may use different strategies. If, for example, a section of an object has been identified by some sensing strategy, but not the segment required for continuation of the Ideal Execution Path, it can be inferred that the object is actually present, and a simple search strategy can be used to find the part of the object required. In the majority of calls of this particular Behavioural Module,

4.4 The Ideal Execution Path and Alternative Execution Paths

Both the Ideal Execution Path and Alternative Execution Paths describe control routes through the hierarchy of Behavioural Modules. The Ideal Execution Path describes the best route, as determined by the system designer, which can be taken through the Behavioural Modules. Alternative Execution Paths are defined if particular Behavioural Modules on the Ideal Execution Path fail to correctly perform their task due to there being too much variation and/or uncertainty in the actual workcell or the strategies being unable to deal with the actual situation in the workcell.

Through any Behavioural Module hierarchy, an Ideal Execution Path can be identified by the designer of the system which reflects the best way in which the assembly can be performed. The criteria for this can be diverse, ranging from time taken to execute the Behavioural Module, to its reliability. In the example in Fig 4–2 the Ideal Execution Path for the task performed by the Behavioural Module, **Demonstration**, can be designated as following the control route through **Behavioural Module 1**, with Exit State BM-1A, and through **Behavioural Module 2**, with Exit State BM-2A.

If the variation and uncertainty in the workcell prove too much for the Behavioural Modules on the Ideal Execution Path, or it is determined that the current Behavioural Module is no longer the most suitable for the task, an Alternative Execution Path may be available to bring the system back to the Ideal Execution Path where execution can continue onward as before. The use of an Alternative Execution Path is not the optimal route that should be followed for the assembly task, but can be useful to prevent failure of the complete system when some alternative action can be performed which could result in successful completion of the task.

Module to determine if a lower level module has completed its task in the required manner for that call. Suppose the existence of a Behavioural Module which finds outlines of objects in an image taken from a vision system. The Exit State from this particular Behavioural Module may be 'Yes, there are one or more objects which I have determined the outline of', or 'No, there are no objects in the image'. Depending on the context of the call of this Behavioural Module, the system may be expecting there to be an object in the image, in which case, if 'Yes' were returned, this would be interpreted as a correct Exit State for that call. If, on the other hand, the Behavioural Module had returned 'No', this would mean that, from the Exit State value, it could be determined that the Behavioural Module had failed to find an object. Even though a correct Exit State is returned from a Behavioural Module, this does not mean that the correct information is bound to be available, due to the generality of the Behavioural Modules. In the above example, the Behavioural Module which calls the lower level object-finding Behavioural Module may obtain the required Exit State for that call, but when a qualifying parameter is examined, only one object may have been found where two were looked for. Thus, although Exit States prove useful in determining if a Behavioural Module has failed completely in its task, even when the right Exit State for the context has been returned, the parameterised information which is returned may prove to be inadequate.

The Exit State plus the range of other returned parameters give an idea of the competence of the Behavioural Module and what it can cope with. Once the competence to perform a task has been constructed from Behavioural Modules, it is possible to find the Ideal Execution Path through the Behavioural Modules by looking at the Exit States and the other control decision points. The following section looks at the ideas behind the Ideal Execution Path and Alternative Execution Paths.

```
move-and-check(position, sensor-state, Exit-State)
      move(position, Exit-State)
      if Exit-State = 'Moved to position' then
           check-sensors(sensor-state, Exit-state)
           if Exit-State = 'Sensors-as-expected' then
                Exit-State = 'Yes'
           else
                Exit-State = 'No'
      else
           Exit-State = 'Not moved'
end
```

Figure 4–3: Example of a general Behavioural Module.

part should be in the gripper, or any particular aspect that the sensors may be able to detect (again, dependent on the situation). The value of the parameter Exit State for the Behavioural Module **move-and-check** is either 'Yes', 'No', or 'Not moved'.

When a Behavioural Module is called upon to perform a task, it is not the problem of the Behavioural Module itself to recognise whether it has succeeded or failed for this particular call, but rather to recognise what has occurred within it and pass this information back up via the Exit State parameter. Exit States are useful for higher level Behavioural Modules to determine what has happened at the lower level, although each level in turn is only concerned with its own internal structure, rather than the interpretation of its output. Once the Exit State has been passed on to the calling Behavioural Module further up the hierarchy, it is up to this Behavioural Module to determine if the Exit State returned is the appropriate one for which the Behavioural Module was called, and also whether the qualifying parameters associated with the Exit State provide valid information for that task.

The following example demonstrates the use of Exit States by a Behavioural

Behavioural Module **Demonstration**. In another section of **Demonstration**, the Behavioural Module **Behavioural Module 3**, produces two Exit States, BM-3A and BM-3B. The first of these links directly to **Behavioural Module 2**, thus not directly becoming an Exit State for **Demonstration**, but the second is converted into the Exit State value of DEMO-B from **Demonstration**.

A simple Behavioural Module may have only one Exit State, while more complex modules, especially those composed of lower level Behavioural Modules, may have many more. The number of values that the Exit State can attain depends on the actual Behavioural Module and its design. Each Behavioural Module has one parameter assigned to carry its Exit State. This means that addition of new Behavioural Modules to lower levels of the system does not affect the number of parameters passed upwards, although the number of Exit State values at each level may subsequently be different. With the addition of another Behavioural Module into **Demonstration**, a new Exit State of DEMO-C may now be needed, requiring only that this value be recognised at a higher level if necessary. The Exit State parameter may also be supported by other qualifying parameters which can provide information required by the task.

The example in Fig 4–3 illustrates a general purpose Behavioural Module which can be called to perform different tasks. As such, the parameterisation of the Behavioural Module and the returned Exit State play an important part in the use of this module. The parameterised values for the Behavioural Module, **move-and-check**, are the position for the robot to move to, the expected state of the sensor after the move, and the parameter which will contain the Exit State. If the robot has been able to move to the correct position, it will return the Exit State, 'Moved to position'. If it has not moved, then the Behavioural Module **move-and-check** will return the Exit State value of 'Not moved'. If the robot did move, the sensors can be checked to see if the **move** Behavioural Module has performed its task correctly, with the sensor state telling the system the expected values, for this call of the Behavioural Module, to be compared with the actual values of the sensors. The sensor states can indicate that a

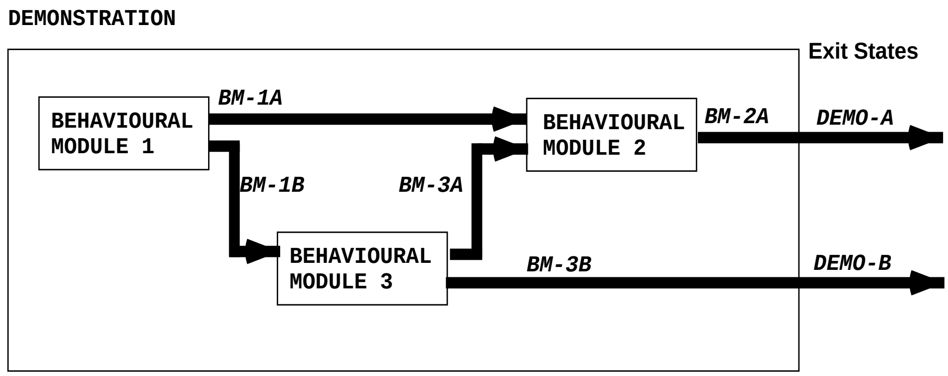


Figure 4–2: An Example of a Behavioural Module.

ceeded or failed while performing the task assigned to it by the system, but only informs the system what it has actually achieved.

The Exit State of a Behavioural Module reflects its internal execution and can be determined from the following:

- The Exit States of lower level Behavioural Modules which are called within the Behavioural Module.

- Sensory or locally available data.

The contents of each Behavioural Module can be partitioned according to its composition. A section can either consist of one other lower level Behavioural Module, or of a section of local robot code, curtailed by some type of control decision where the control path divides when a decision is made. This decision can be taken from sensor data, or other local information available at that point. At the end of each section of code, an Exit State value for the Behavioural Module may be produced to distinguish the difference in control path through it. Whether the section produces one or more Exit State value from the Behavioural Module or not depends on the decision of the designer. In the example Behavioural Module diagram in Fig 4–2 (which will be referred to throughout the remainder of this chapter — see Fig 4–1 for the key for this diagram), the Behavioural Module **Behavioural Module 2** produces its own Exit State BM-2A, which is converted into an Exit State of DEMO-A for the

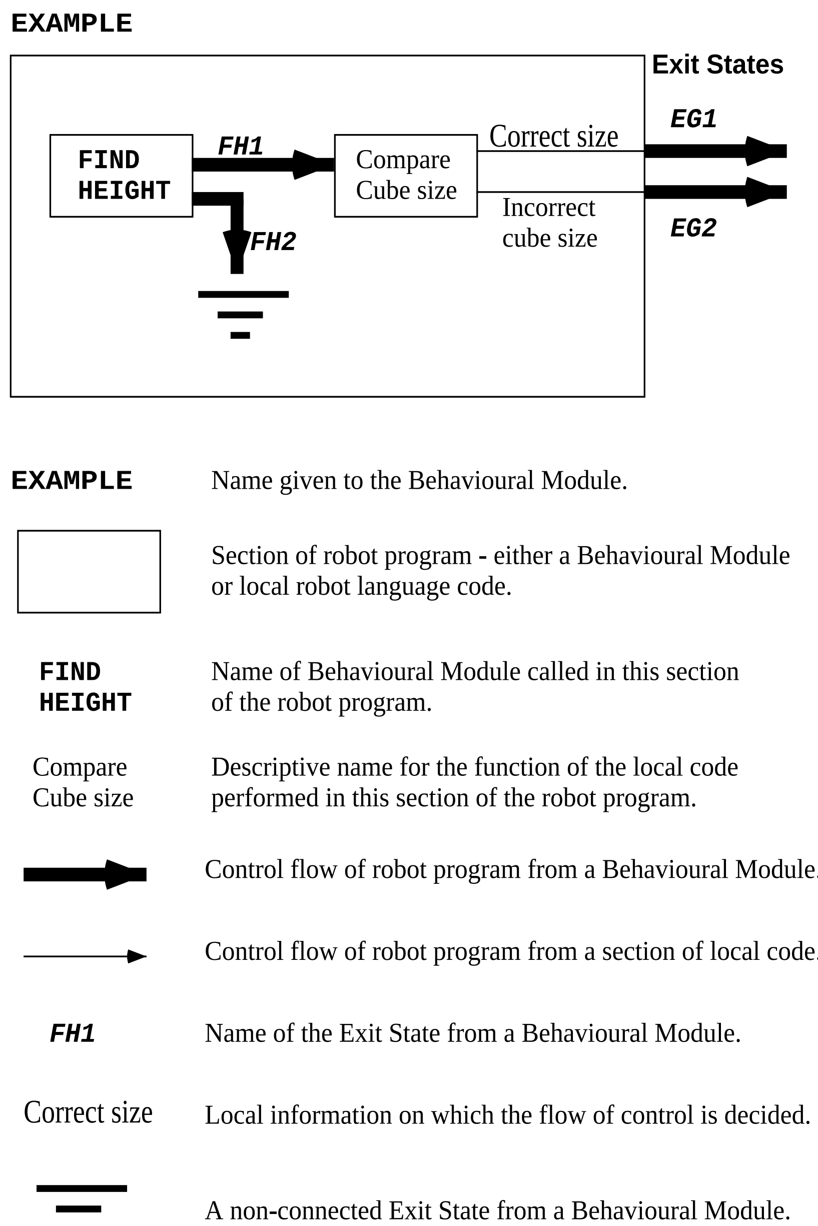


Figure 4–1: Key for Behavioural Module Diagrams.

correctly within the context of its calling. If it has not, an alternative series of actions may be initiated to attempt to prevent the system from falling automatically into an error situation. This type of system relies on real world testing of the Behavioural Modules as they are developed and using the context in which the Behavioural Modules are actually called to determine suitable alternative actions.

The following sections look at the diagrams and terminology which has been designed to help with the introduction of the ideas above: Exit States, the Ideal Execution Path, and Alternative Execution Paths.

4.2 Diagrams

Before continuing with this chapter, the diagrams which will be used for the rest of the thesis must be introduced. Fig 4–1 shows an example of a diagram which illustrates a Behavioural Module called **Example**. This Behavioural Module is composed of a lower level Behavioural Module, **Find Height** and a section of local robot language code. This figure should be referred to when other similar diagrams are encountered. The key explains the labeling conventions for each of the symbols which are used, and an example of each symbol is incorporated in the diagram.

4.3 The Exit State

This section looks at the concept of Exit States and examines how they are determined, what they are, and what they can be used for.

The Exit State of a Behavioural Module provides an indication of the internal control path which a particular call of that Behavioural Module has followed. This value does not determine if the Behavioural Module has suc-

The chapter continues by examining how the above ideas can be used, then concludes with a summary.

4.1 System Reliability

This thesis examines a system which was built to investigate the integration of software and hardware to ensure that the Behavioural Modules can deal reliably with the tasks they perform, and to investigate the linking of these Behavioural Modules in a reliable way. To do this, it was necessary to introduce sensing into the system.

In Chapter 2, it has been mentioned that sensors can perform two basic functions when integrated into an assembly system: they can gather data to change the destination of the robot arm, or they can change the sequence of actions of the robot. Both these uses can increase the reliability of the assembly system. Although this work looks in passing at several strategies which use sensors to guide the destination of the robot arm, it is mainly the change of the sequence of actions which will be examined here.

Sensors can be used to monitor Behavioural Modules to detect what is occurring during their execution. The hierarchical structure of the system allows sensors to be integrated at any level, allowing the flexibility to monitor actions where necessary. Therefore, the real world can be used as much as possible to provide immediate information for the system, rather than relying on data from a composed internal world model. The close integration of hardware, software and the task can be achieved at all levels of the system and this allows the Behavioural Modules to be specifically adapted to the task. This does not limit the generality of the whole system as long as the tasks performed by the Behavioural Modules are general over the assembly domain.

The above monitoring of the Behavioural Modules, along with information obtained from the planner and instantiated at run-time with real world data, allows the system to determine whether a Behavioural Module has performed

Chapter 4

Increasing the Reliability of Behaviour-based Robotics

Chapter 3 has introduced a Behaviour-based robotic assembly system which provides a simplified planning system connected to a competent execution system. The reliability in the system is achieved by designing Behavioural Modules which can individually deal with the variation present in the world and attempt to reduce its effect by using motion strategies.

This thesis examines a method of increasing this reliability by integrating sensors which help to provide reliable linking between Behavioural Modules. This allows for a more responsive system which can deal with Behavioural Modules which are unable to successfully complete their task.

The following chapter outlines the methodology and terminology which is used to describe the thesis work. Beginning with a general introduction to the ideas, the chapter continues with an introduction to *Exit States*, the *Ideal Execution Path*, and *Alternative Execution Paths* which together form the basis for the reliable linking of the Behavioural Modules in the assembly execution system. The Exit State provides information on how a Behavioural Module performed its task. The Ideal Execution Path and the Alternative Execution Path describe control routes through the Behavioural Module hierarchy, which can be used to prevent the assembly system falling into an error situation.

been mentioned. As parts are put into the assembly, the planner makes sure that each downward facing cube is supported either by the workcell floor or by some previously inserted cube. This also means that no overhangs are permitted in the assembly, so there are some shapes which cannot be planned for with this planner. Another method in the workcell for ensuring no collisions is to use the idea of transporting all parts at a pre-defined safe height. After picking up a part at any stage of the assembly, it is lifted to a safe height in the workcell and transported at this height. The robot also moves at this safe height when not burdened by a part.

The SOMASS system provides a demonstration of how reliability can be achieved by appropriate modularisation of the task and dealing with variation within individual Behavioural Modules. The work in this thesis looks at enhancing the system by the introduction of sensors and investigating how the reliability can be improved by appropriate linking of the Behavioural Modules using information from these. Chapter 4 introduces the terms and ideas for the thesis work.

include the integration of a vision-guided part acquisition Behavioural Module [Chongstitvatana & Conkie, 1990]. These involve no significant changes to the planner.

3.2.5 SOMASS Reliability Strategies

This section provides a summary of the ways in which the SOMASS robotic assembly system deals with the variation in the workcell and parts to provide a reliable system. Uncertainty is not dealt with in this system as sensors are not employed (see Chapter 2, Section 2.1 for the definition of the word 'uncertainty' as used in this thesis). The management of variation is dealt with in two ways in the SOMASS system. The first is in variation-reducing motion strategies, and the second, by introducing rules constraining certain parts of the assembly.

All the variation-reducing motion strategies attempt to use the robot to manipulate the assembly object by some means such that, at the end of the motion, the part is in a more accurate position. The two examples of this are the sweeping of the parts into place before picking them up, and using a double snap to centre the part in the gripper before lifting it. Both of these manipulate the workpiece in such a way that the variation which may have been present before the motion is very much reduced. A useful, but less generally applicable example of the robot itself being used as a variation-reduction implement, is the final patting together of the cube. This is the climax of the variation containment strategy which prevents collisions with other parts on put down by leaving gaps in the assembly. These are squeezed out by patting the cube together at the end.

The second way of producing a robust assembly relies on general rules and strategies invented by the designer. Constraining the system such that parts can only be acquired in a downwards direction ensures that the gripper does not collide with other parts or the workcell floor. The same is true for the rule stating that the parts are only put into the final assembly in a downwards direction. The previous strategy of leaving pads in the assembly has already

calculates the x and y offsets needed for each part to safely insert it in the assembly. In the actual Behavioural Module, the distance of the gap is calculated at run-time according to the actual size of the cubes used. (See Fig 3–11.)

The combination of the above Behavioural Module which leaves spaces, and a Behavioural Module which pats the shape back together, shows an example of a variation-reduction strategy performed over several modules. The **zvpatcube** Behavioural Module only works for the 3 x 3 x 3 cube shape. The sweeping implement is applied to the four sides of the assembled cube in turn, gradually moving the parts back to their proper place in the assembly. Two rounds of the assembly are normally made. The padding of the parts cannot be used with certain Soma-5 shapes where cyclic adjacencies occur.

These basic Behavioural Modules produce a reliable assembly system. Reliability tests of the system have been performed and these are reported in [Malcolm & Smithers, 1988]. In 517 runs involving large and small wooden Soma-4 sets, and assembling 12 different assemblies, only 12 failures occurred, mainly due to previously undiscovered bugs in the planner. Timings for the planner are also reported in that paper.

The SOMASS system, in spite of its lack of external sensors, can still perform reliable assemblies in the real world. In acquiring a Soma part, a large amount of rotational and locational variation can be dealt with, and the padding strategy allows much variation on part placement. Due to built in simplifications, certain assemblies cannot be built, in particular, those with any overhanging cubes, those which need part placement to allow for finger clearance on more than the last part, and those tall enough to be unreachable by the robot using the strategy of downward insertion. If anything does go wrong in the assembly process, the robot has no knowledge of this and will continue regardless with its pre-programmed actions. If some part of the assembly fails, the whole assembly will fail.

A later version of the SOMASS system (implemented by Petropoulakis) incorporates a different regrasp strategy which can deal with the regrasp table being inaccessible from two sides. Later developments of the SOMASS system

Regrasp

If, due to the use of the five degree of freedom robot, the Soma shape cannot be directly inserted into the assembly using only a z rotation from the original orientation on pick up, the **zmanip** Behavioural Module is used. This Behavioural Module is one of the most susceptible to failure due to variation in the environment as no attempt is made to perform any variation-reducing strategies, although such strategies are possible, such as sweeping the parts back into place as before. The planner decides if a regrasp must be performed, and, if so, provides information about the cube to be regrasped and the orientation of the gripper and the wrist for the grasp. All these are passed to the module by parameters. The picked up part is transported to a small regrasp table, where the gripper can now approach the part from the side. This could not have been done on the original pick up due to the possibility of wrist collision with the top of the worktable. The part can be placed on the regrasp table, then picked up again from another orientation, and perhaps by grasping a different cube. No variation-reduction strategies are introduced in this Behavioural Module, so, although the parts are initially held in a known position in the gripper from the pick-up Behavioural Module, the uneven surface of the regrasp table occasionally causes the parts to rock and lose their position, or the part can slip in the gripper while being turned. When the part is picked up from the regrasp table, it may no longer be centred in the gripper.

Part Placement

The final put down Behavioural Module, **zput** is facilitated by a variation containment strategy which involves leaving gaps in the assembly. If the previous Behavioural Modules, especially the **zmanip** one, do not hold the part correctly, have not picked it up correctly, or it has slipped slightly in the journey through the workcell, the part may collide with the already completed partial assembly, causing failure of the assembly. This may also happen due to inaccurately made cubes. To remove this possible source of failure, the planner

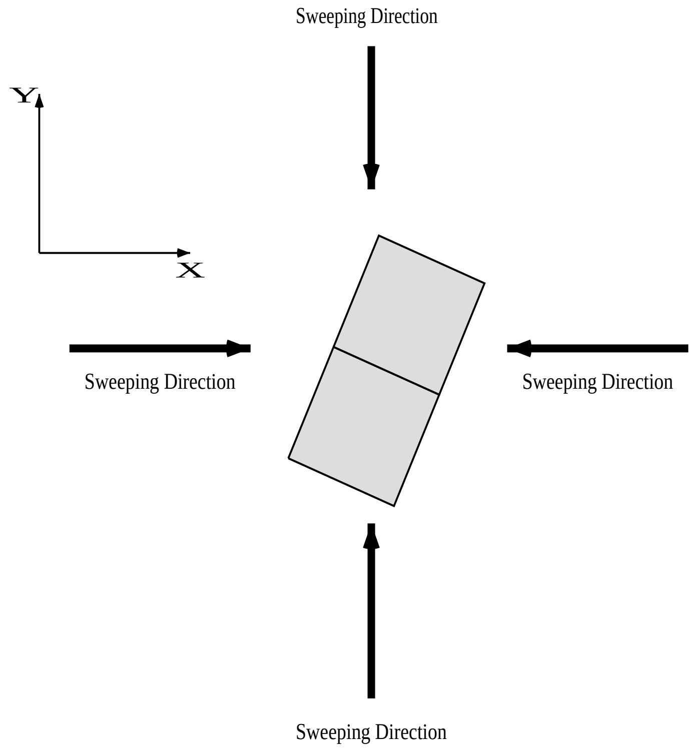


Figure 3–15: Strategy to centre the Soma part by sweeping motions.

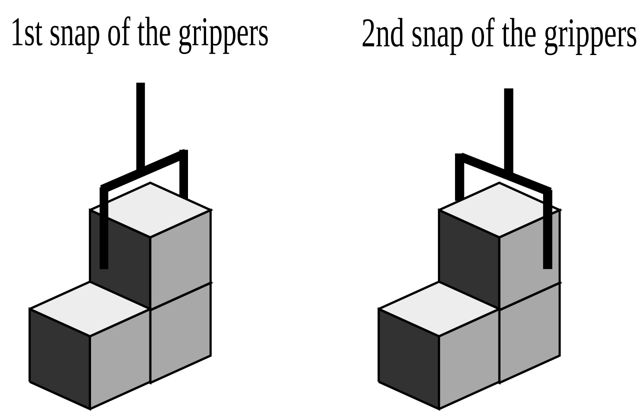


Figure 3–16: Strategy to centre the Soma part by a double snap.

- *location* — taught location.

zvpatcube (location)

- *location* — taught location.

The main three stages of the assembly, namely 'Part Acquisition', 'Regrasp' and 'Part Placement' are described in more detail in the following sections.

Part Acquisition

The Behavioural Module designed to pick up the Soma parts is split into two sections. See Fig 3–15 and Fig 3–16 for diagrams representing the two motion strategies. The first of these, **zpatget**, centres the Soma shape by using a straight piece of wood to sweep the part from all four sides. This reduces the variation in the initial placing of the shape in the workcell and allows the gripper to move down over the part without fear of colliding with the top of the cube. The sweeping strategy is quite restrictive in that a lot of space is required around the parts, and some of the Soma-5 parts swing round to the wrong acquisition orientation on sweeping due to their awkward shapes and the friction of the part on the table.

Once the part has been centred by sweeping, there are still two possible methods of actually grasping it. The first, **zget** (see Fig 3–16) is used if a single upstanding cube can be grasped. In this case, the gripper is turned through 90 degrees around the z axis from the grasp orientation provided from the planner and a single snap of the fingers is made. This centres the part effectively in the gripper in this direction. The gripper then picks up the cube from the normal orientation. If, as with the 'zed' part, two or more cubes are present and the double snap cannot be performed, a single grasp is made, using the **zpick** Behavioural Module. The single grasp, **zpick** is also used for picking up the wood for the sweeping motions.

zpatget (location, xrot, xminus, xplus, yrot, yminus, yplus)

- *location* — central taught location around which the sweeping motions occur.
- *xrot* — rotation around the z axis from normal location needed to perform a sweep along the x axis.
- *xminus* — offset from the central location to account for the shape of the part. Part swept up to this distance from centre.
- *xplus* — as for xminus (2nd sweep).
- *yrot* — as for xrot, but to align for sweeping along the y axis.
- *yminus* — as for xminus (3rd sweep).
- *yplus* — as for xminus (4th sweep).

zpick (location)

- *location* — taught location.

zget (location)

- *location* — taught location.

zmanip (location, putrot, pxn, pyn, pzn, getrot, gxn, gyn, gzn)

- *location* — taught location for centre of regrasp table.
- *putrot* — rotations from normal location for depositing part.
- *pxn, pyn, pzn* — x, y and z offsets for the grasped cube to place the part in a stable position on the regrasp table.
- *getrot* — rotations from normal location for reacquiring the part.
- *gxn, gyn, gzn* — x, y and z offsets for the new cube to be regrasped as it sits on the regrasp table.

zput (location)

```
.PROGRAM zget(pos)
;
; sets robot speed, approaches at a safe height above Soma part
; to be grasped, opens gripper, approaches part at a 90 degree
; rotation, moves down, opens/closes gripper, departs then moves
; down in correct orientation to grip the part and depart to a
; safe height.
;
        SPEED transp
        APPRO pos, clear
        OPEN
        APPROS pos:TRANS(0, 0, 0, -90, 0, 0), approach
        MOVES pos:TRANS(0, 0, 0, -90, 0, 0)
        CLOSEI
        OPENI
        DEPARTS approach
        APPROS pos, approach
        MOVES pos
        CLOSEI
        DEPARTS clear
        RETURN
.END
```

Figure 3–14: VAL-II Behavioural Module — designed to pick up a Soma part.

assemblies and each Behavioural Module is designed to cope with a certain amount of variation in the workcell, depending on the particular strategies employed. Strategies were also implemented over several different Behavioural Modules to help to produce a reliable system. An example of this type of strategy is found in the patting strategy on page 82.

The Behavioural Modules know nothing of the assembly order, the space occupancy of the parts, or the final assembly position. These are all passed to them by parameters given in the plan in terms of relative information and are instantiated with real workcell data at run-time. The system at this point knows the nominal initial locations of the parts, the nominal cube size and several placing locations, such as a location which defines where the final assembly should be built. It is assumed that the Soma shapes are presented in an initial nominal configuration, within an allowable translational and orientational variation. These workcell details are required to ensure that there is no interference between the starting locations for the parts and the other activities in the workcell. The initial locations for the parts must be well spaced due to the pick up strategy employed, providing a problem of workcell overcrowding if more parts are introduced. This could be reduced by feeding the parts in individually by another robot, or using a feeding mechanism. The initial configuration was also designed so that the shapes would be gravitationally stable and presented with a single upstanding cube to be grasped. This was not possible with the 'zed' part which was presented with two topmost cubes.

Behavioural Module Implementation

The main Behavioural Modules, with associated parameters provided by the planner, are listed in the following description. Each location mentioned refers to a VAL-II defined location in the world coordinate frame, given as (x, y, z, yaw, pitch, roll) and taught to the robot using a teach-pendant. Locations are treated as distinct from positions or points which are defined as [x, y, z] only. The rotations are rotations from a given normal location. An example of the actual VAL-II code for the **zget** Behavioural Module can be found in Fig 3–14.

```
; ----------- The placing of right -----------
    CALL zpatget(b7.get, RZ(90), -1.5,0.5, RZ(0), -0.5,1.5)
    CALL zget(b7.get:RZ(0))
    CALL zmanip(table, RZ(0):RY(270),-1.0,0.0,2,RZ(0),0.0,0.0,2)
    CALL zput(b7.put:RZ(0))
```

Figure 3–13: VAL-II pre-coded procedures generated from the Prolog list.

3.2.4 The SOMASS Behavioural Modules

The on-line section of the SOMASS system is performed by Behavioural Modules which execute the plan specified by the planner. Written in VAL-II, these Behavioural Modules run directly in the robot controller and are activated by procedural calls from the plan created during the planning process. They are structured in a hierarchical manner and realise the part motions which are given in the plan. The top level of the hierarchy deals with the Behavioural Modules **zpatget** (centre the part for pick up), **zpick** (pick up the part with a single snap of the gripper), **zget** (pick up the part with a double snap of the gripper), **zmanip** (regrasp the part if necessary), **zput** (place the part in the assembly), and **zvpatcube** (push the final assembly together). Fig 3–13 shows a typical call produced from the planner of a sub-plan to place one part into the assembly. Lower level Behavioural Modules can be called in turn from these. The actual Behavioural Modules used are described on page 76.

Behavioural Module Characteristics

In the early version of the SOMASS system which is described here, no sensors or sensory data were used, so it was only the variation in the workcell and parts which had to be dealt with to achieve a reliable assembly system. The system relies on variation-reducing motion strategies to produce these reliable

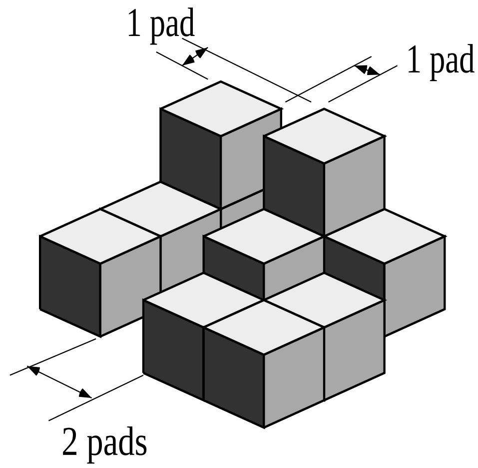


Figure 3–11: Padding of assembly.

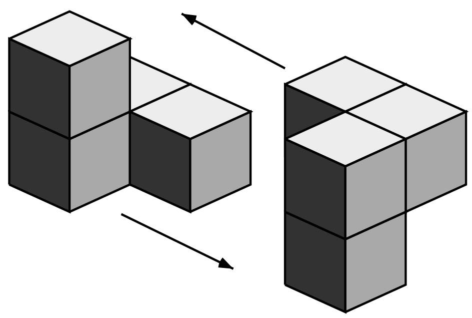


Figure 3–12: Cyclic adjacency relationship of parts.

With the Soma-4 set, this offset is always possible, but when the Soma-5 set is used, or even duplicate parts in the Soma-4 set, cyclic adjacency relationships may occur where the parts cannot possibly be separated in this manner (see Fig 3–12).

Once the final Prolog list has been completed, containing the information shown above, it is passed through the post-processor to provide a set of sub-plans in VAL-II. Fig 3–13 is an example of part of the plan produced in VAL-II code by the planner to pick, regrasp, and place a part. It shows the actual VAL-II code generated for the part, named 'right', which was produced from the Prolog list previously described.

The next section looks at the Behavioural Modules which are called from these sub-plans.

parts which later cannot be filled due to the rule of parts always being inserted vertically downwards.

3. **Put-down Grasp** This section of the planner determines if a valid put-down grasp can be obtained for each part. There must be clearance for the gripper fingers to open to release the part in the actual assembly. Thus there must be no other cube already in place to the sides of where the gripped cube will be placed in the assembly. This rule can be relaxed in some cases where the last part cannot be inserted in this manner due to the shape of the assembly. In the 3 x 3 x 3 cube, for example, it is impossible to insert the last part in this manner. Instead, the part is placed offset from the put down location. The wrist position is automatically clear of the assembly as the parts are inserted vertically downwards and no part is higher in the assembly than the one being inserted.

4. **Regrasp Planning** When the part cannot be placed directly from the initial starting location into its final place in the assembly using only rotation around the z axis, the planner checks to see if the part can be regrasped. Alternative grasps for the part are examined to find any that may be successfully performed by the robot, taking into account that the robot used to perform the assembly has only 5 degrees of freedom. Again, backtracking takes place if a regrasp cannot be found.

After the planner has found a solution which can be performed in the real assembly workcell, an addition is made to help the put-down variation-reduction strategy. This involves calculating a small distance, or pad which is left between the parts on put-down (see Fig 3–11). The actual distance left is calculated at the run-time of the actual assembly, and is dependent on the size of the cubes. The information in the planner is in terms of unit offsets. After one part is placed, the offset for the next is calculated. This is in terms of an x and y offset, with each axis treated separately. If a previous part has been offset, the next offset may be more than one pad as the chaining effect of other faces may dictate that more offset is necessary.

Certain heuristics can be used to speed up the planning process, such as looking at the rotations of parts which correspond with their initial configuration first. In the Soma-4 set, choosing the part with only three cubes to be inserted first allows the remaining unfilled spaces in the assembly to be partitioned into multiples of four to see if the remaining parts can possibly be fitted in.

The planner attempts to find a general solution to the problem of fitting the parts into the assembly, then decorates this solution with details required to perform the actual assembly, using backtracking between the different stages of the planning. A complete plan is formed at each stage, before being expanded at the next level. There are four stages of the planner, with each backtracking to the previous on failure, except in some special cases where failure-directed backtracking is used to miss out obviously impossible trials.

1. **General Solution** A general solution for an assembly is one way in which the parts will correctly fit into the final assembly. For the Soma-4 shapes and the 3 x 3 x 3 cube, the parts can fit into the final assembly in 240 different ways excluding rotational and mirror symmetries. The planner finds 1440 solutions as gravity is important for this problem. The parts can be rotated and translated from their initial starting orientation. These movements may not be physically possible in the real assembly cell, but such details are examined at later stages of the planner. Each part is placed in turn, then the next part compared with the remaining spaces in the assembly, backtracking when failure is encountered.

2. **Gravitationally Stable Ordering** Having found a general solution for the parts, an attempt is made to find an ordering of the assembly which will allow each part to be inserted vertically downwards into the already completed subassembly, and come to rest in a gravitationally stable position. This means that each downward surface of the part must be supported either by the workcell surface, or by a cube of an already inserted Soma part. This has the side effect of ensuring that no gaps are left under

Figure 3–9: Regrasp placement of the 'right' Soma-4 part.

Figure 3–10: Regrasp acquisition of the 'right' Soma-4 part.

- **Regrasp Details**
  - [[[**z**,**0**],[**y**,**3**]],

    Rotation orientation of wrist for placing part on regrasp table.
  - [[**0**,**0**,**1**],[**1**,**0**,**0**],[**1**,**0**,**1**],[**1**,**1**,**0**]],

    Orientation of part on the regrasp table.
  - [[**1**,**0**,**0**]],

    Offset of base of part relative to table centre.
  - [[**0**,**y**,-(**x**)],[**0**,**0**,**1**]],

    Axis, wrist orientation and gripped cube for release.
  - [[**0**,**y**,**z**],[**1**,**0**,**1**]],

    Axis, wrist orientation and gripped cube for regrasp.
  - [[**z**,**0**]]],

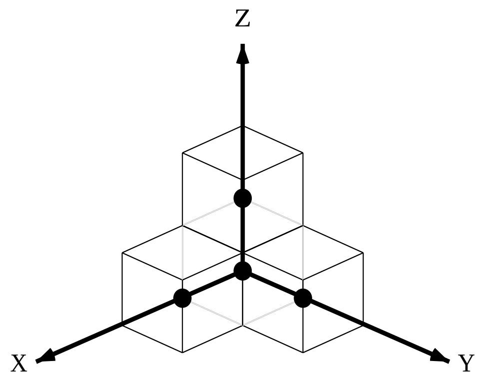
    Rotation of wrist for eventual placing in assembly.
- [**1**,**0**],

  x and y padding offsets for the part in the final assembly.
- [[[**z**,**0**],[**y**,**3**]],[[**0**,**0**,**0**],

  The rotation and translation through which the part must be moved relative to its initial location in order to place it in the final assembly.
- [[[**0**,**0**,**1**],[**1**,**0**,**0**],[**1**,**0**,**1**],[**1**,**1**,**0**]],

  The list of ordered points stating the final location relative to the assembly origin of where the part will be located in the final assembly.
- [[**0**,**y**,**z**],[**1**,**0**,**1**]]]]]]

  The axis along which the gripper and the wrist should be aligned on put down of the cube, and the cube to be gripped.



[[0, 0, 0] , [1, 0, 0] , [0, 1, 0] , [0, 0, 1]]

Figure 3–8: Representation of the 'fork3' Soma-4 part.

'right' of the Soma-4 set, integrated with a description of what each piece of information represents. The wrist and gripper orientations and part positions on the regrasp table are shown in the photographs in Fig 3–9 (regrasp part placement) and Fig 3–10 (regrasp part acquisition). The full Prolog list from which this was extracted can be found in Appendix A.

- [**right**,

  The identifying name for the Soma shape which is also used as a reference to its approximate taught location in the robot cell.

- [[[**0,0,0**],[**0,1,0**],[**1,0,0**],[**1,0,1**]],

  A Prolog list of ordered points stating the initial orientation in which the part should be presented in the workcell (taken from a [0,0,0] origin for the part).

- [[**0,y,z**], [**1,0,1**]]],

  The axis along which the gripper and the wrist should be aligned on pick up of the cube, and the cube to be gripped.

3.2.3 The SOMASS Planner

The planner is designed to deal with the Soma shapes and assemblies, and, as such, incorporates several features particular to this world. In the original SOMASS system, only the Soma-4 shapes were used in the assemblies. Written in Prolog [Clocksin & Mellish, 1984], a language which naturally lends itself to working with lists, the planner uses this facility to produce a file containing information to perform the assembly, written in VAL-II. The aim of the planner is to provide a plan which can be used to assemble the seven Soma-4 pieces into a specified assembly pattern. This system was designed to be easily expanded to include planning for the Soma-5 set at a later stage.

The planner was designed to have as little knowledge of the real world and assembly domain as possible, leaving the workcell details to the execution system. The input to the planning section of the system consists solely of a representation of the shapes of the parts themselves, and a similar representation of the shape of the final assembly. For ease of use, and simplicity of drawing diagrams to understand the output from the planner, each cube was shrunk to an [x,y,z] point, and the Soma shapes represented as a list of these points. For example, the four cubes in the 'fork3' part are described by the representation, [[0,0,0],[1,0,0],[0,1,0],[0,0,1]] (see Fig 3–8). The final assembly is described in a similar fashion. The planner can now make use of the simplicity of the integer numbers to manipulate the data since only simple arithmetic operations are now necessary to calculate part rotations, gravitational stability, and space occupancy. These may be replaced by more complicated procedures if different assembly pieces are used.

The planner is hierarchical and makes use of Prolog's backtracking facilities to search the space of possible solutions for the particular assembly. The output from the planner contains a generated list which is post-processed into VAL-II. A sub-plan is produced for each shape, describing how it can be acquired, re-oriented if necessary, and set down in the assembly. The following shows an example Prolog sub-plan for a particular pick, regrasp and place for the part

placed within the translational bounds of the part acquisition sweeping motion (see Section 3.2.4, page 79) and must be within an orientation of approximately $\pm$ 40 degrees of the nominal initial position. The limitation of the gripper opening distance means that the parts must be picked up across a single cube length. This, along with a rule which states that a part must be grasped vertically downward, leads to a simplification rule that states that the parts must be initially placed, if possible, with a single topmost cube (the Soma-4 set shown in Fig 3–2 and Fig 3–3 shows these positions). In the Soma-4 set, this is possible with all parts except for the 'zed' shape. This is unstable in all positions with a single upright cube, and must instead be presented with two cubes uppermost.

For insertion of the Soma shapes into the final assembly, again, it is insisted that a vertically downward motion be used. This helps to avoid collisions with other parts which have already been inserted. An allowance must also be made for finger clearance on opening the gripper to release the shape. Thus, there must be space on either side of the gripped cube of approximately half a cube length. This cannot be done for the last piece to be inserted into some assemblies, such as the 3 x 3 x 3 cube, so the part is set down in an offset position.

When the parts are placed into the final assembly, gaps are left, determined by the planner, to ensure that even when the parts are subject to variation, they can still be inserted into the assembly without knocking into others already there. This is called the strategy of *padding* the assembly. Once all the parts have been assembled into their padded positions, the final assembly can be pushed together to reduce the gaps (called *pads*). This process is referred to as *patting*. [Salmon, 1989] implemented a patting strategy which dealt with most assembly shapes.

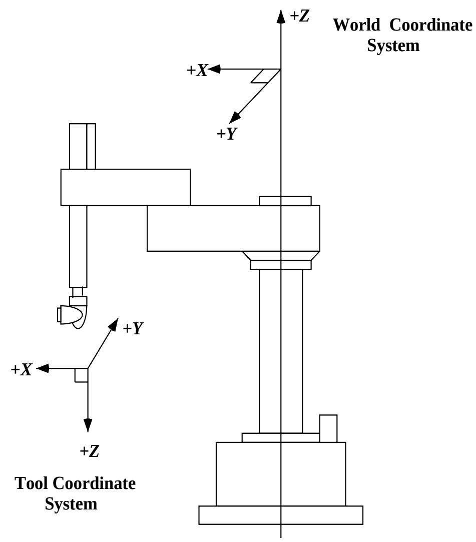


Figure 3–7: The five degree of freedom Adept Robot.

components of the robot hand orientation at that point. Direct movement to a specific location can be made, or movement can be made relative to the current location, either by straight line motion, or through a joint-interpolated path. Special commands have been inserted to access sensor data, although in the first experiments with this system, sensors were not employed.

The planning component of the system runs on a SUN3 workstation, which produces a file of VAL-II commands which are, in turn, passed on to the Adept controller via an RS232 link. The Behavioural Modules, stored in the Adept controller, are programmed in VAL-II and called directly by the plan.

The planner and the execution system both make use of some explicit knowledge of the task and the workcell. Some rules of assembly were defined which were the consequence of using a five degree of freedom robot, and also some which were included to make the assembly more reliable. Some of these rules are listed in the following paragraphs.

In the initial workcell configuration, the Soma-4 parts are placed in stable configurations, with adequate space between them to avoid collisions due to robot motion strategies when acquiring the parts. The parts must also be

creased. The physical parts used in the assembly experiments were constructed from roughly made wooden and plastic cubes, inaccurately glued together, which often gave the shapes a tilt when set on the work surface. Therefore, the problem of variation was still present due to the differences from the ideal of the individual shapes, and the propagation of the effects of these differences as the shapes were introduced into the assembly.

3.2.2 SOMASS System Overview

The SOMASS system was designed as an experimental investigation into the Behaviour-based approach to robotic assembly. The system, composed of a symbolic planner and a Behaviour-based execution system, is linked by the plan itself, which has important implications on the architecture of the system. The aim of the project was to show that appropriate Behaviour-based task decomposition into a competent run-time system composed from Behavioural Modules could produce a good foundation for robotic assembly. It was also intended that the system show that Behavioural Modules could be used to build useful assembly systems and provide a good interface to a simplified planner. The planner and Behavioural Modules were designed to deal with assemblies in the Soma world, using Soma shapes as the building components and some of the many possible assemblies as the final assembly goal. The ideas were tested in the real world in an assembly workcell with its associated variations and uncertainties (see [Malcolm & Smithers, 1988]).

A five degree of freedom Adept robot provides the central feature of the experimental workcell and can be seen in Fig 3–7. The Adept robot can be programmed using the VAL-II robot programming language [Shimano *et al.*, 1984]. VAL-II supports certain well known program control statements, such as conditional and simple loop structures, but also includes specific commands for the control of the robot. The robot can be directed to a location in the workcell. Location is defined, in this case, as being a combination of (x, y, z) position relative to a specified coordinate frame, and the yaw, pitch and roll

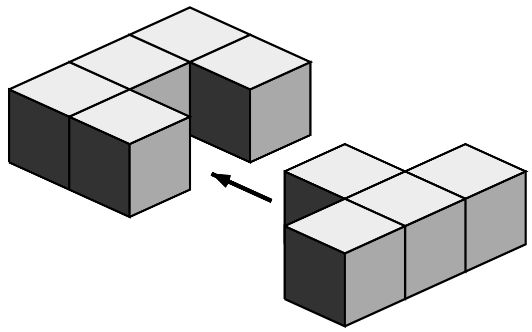


Figure 3–5: An example of the 2D Peg-in-Hole in the Soma-5 world.

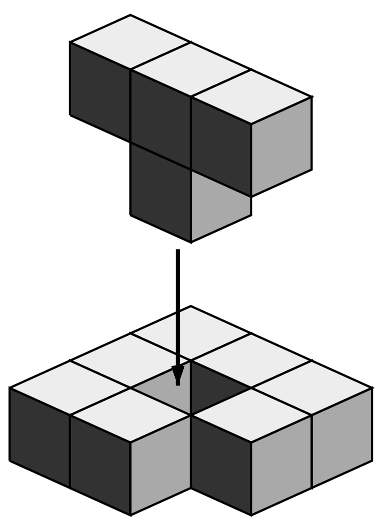


Figure 3–6: An example of the 3D Peg-in-Hole in the Soma-7 world.

Soma Domain for Assembly

The Soma world was selected as the domain in which to perform experiments in assembly and test an integrated planning and robotic assembly system.

The requirement for this kind of system was a simple experimental domain which still retained the important properties of assembly tasks, such as the fitting of parts together, without the geometric complexity normally associated with such tasks. The assembly of the Soma shapes involves shape-dependent part fitting. The Soma shapes are geometrically simple, thus making it easy for a planner to represent and reason about them, yet they are not simple 'blocks world' objects, whose assembly properties consist only of 'placing on' or 'next to'.

Planning for this particular kind of assembly involves looking at the vast combinatorial problem of space occupancy while the actual assembly by robot involves placing the parts into an assembly, constrained by the part and assembly shape. Thus the geometry of the parts is simplified, but the planning problem is still complex. In an industrial setting, as with this system, the part placement would be worked out off-line and given to the run-time system. Humans, with their advanced sensory systems and dexterous fingers, find it easy to manipulate the physical Soma shapes, but find it difficult to cope with the large solution space of the assembly problem. It is relatively easy for a computer program to search through the solution space, but the physical assembly provides problems for a robot, which has a greatly limited sensory capability.

The Soma World is extendible, with the Soma-4 set easily expanded to the Soma-5 set (see Appendix C for a spine representation of the Soma-5 set). New problems are encountered with each expansion, as well as retaining the old ones. An example of this is the 2D peg-in-hole problem which initially arises in the Soma-5 set (see Fig 3–5), but the 3D peg-in-hole problem only appears when the Soma-7 set is introduced (see Fig 3–6).

The geometrical properties of the Soma shapes are simplified, but this does not mean that the physical assembly problem of variation in tolerances is de-

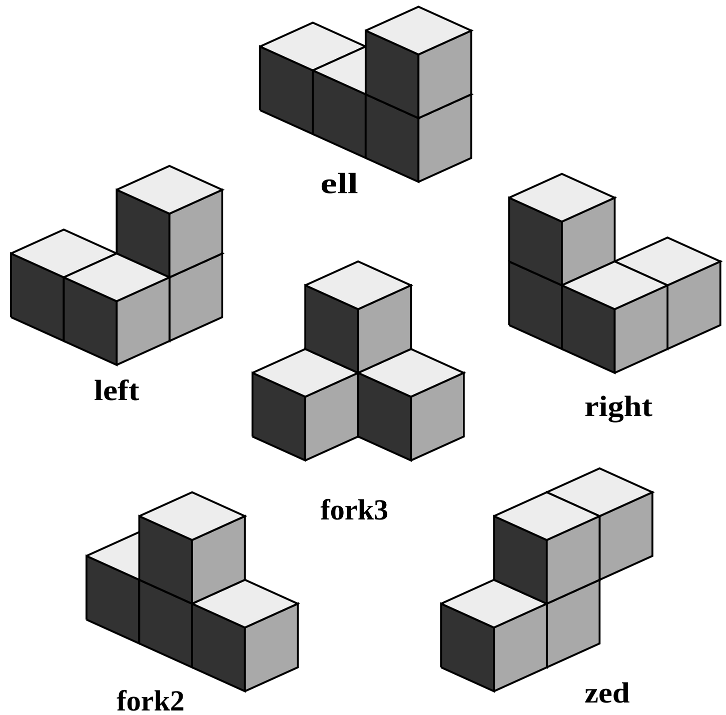


Figure 3–3: Soma shapes composed of 4 cubes.

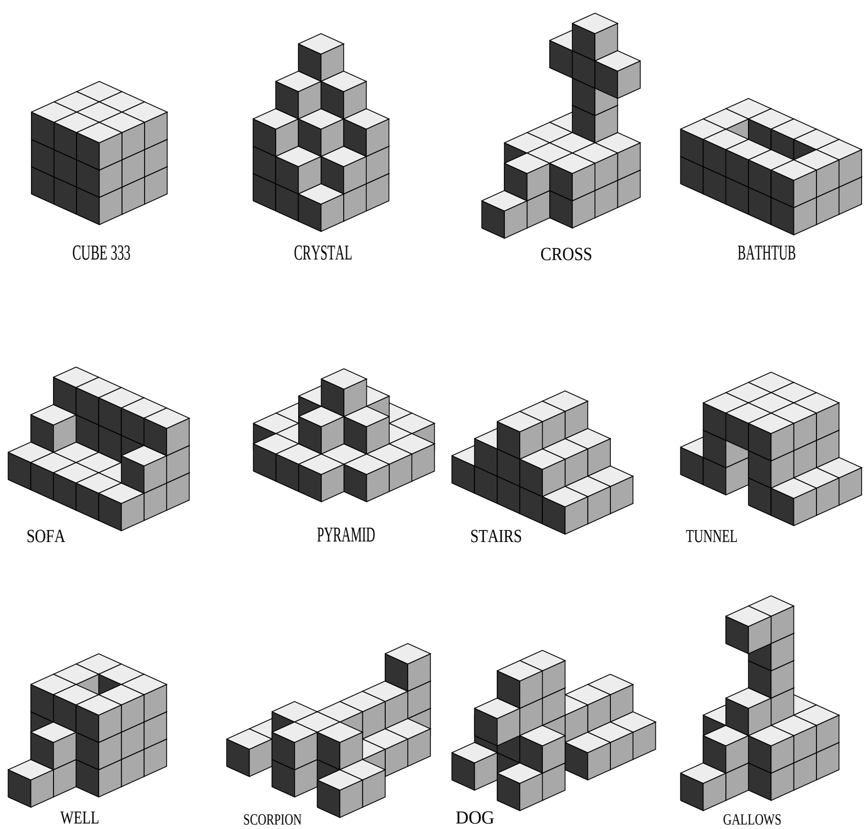
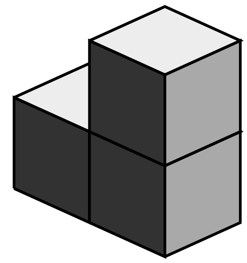


Figure 3–4: Some Soma Assemblies.



lell

Figure 3–2: Soma-3 set.

3.2.1 The Soma World

The Soma Shapes

Soma shapes are built from individual cubes which are joined together according to certain construction rules. Each cube can be squarely joined by one or more of its faces to other cubes of a similar size, to form shapes which are irregular, that is, with at least one surface concavity. The only possible arrangement for two cubes is obviously not concave, and therefore not a Soma shape. The first Soma shape, and, incidentally, set, comprises three cubes as seen in Fig 3–2. This is referred to as the Soma-3 set, the set of Soma shapes with three or fewer cubes.

There are six instances of Soma shapes with four cubes (see Fig 3–3). These are combined with the one Soma-3 shape to form the Soma-4 set, the set of Soma shapes with four or fewer cubes. This can be expanded to include shapes with up to 'n' number of cubes, to form the Soma-n set, as long as they conform to the above rule of irregularity.

The Soma-4 set, as a puzzle, was first discovered by the Danish Mathematician, Piet Hein around 1936 [Gardner, 1961]. It can be shown that the seven Soma-4 shapes can be put together to form a 3 x 3 x 3 cube in 240 different ways, excluding rotational and mirror symmetries [Berlekamp *et al.*, 1982]. Many other assemblies are also possible using the seven shapes from the Soma-4 set. (See Fig 3–4.)

3.2 The SOMASS Robotic Assembly System

The SOMASS assembly system, designed and implemented by Malcolm [Malcolm, 1987], provides a complete planning and execution system which performs the assembly of Soma shapes. Developed as a first experiment in building a Behaviour-based robotic assembly system, it was designed to explore the representational and architectural problems involved. High level descriptions of the task are used by a symbolic planner to produce plans which provide an interface to the actual run-time execution system.

The SOMASS system is divided into two parts: the planning system and the execution system. The planner does no explicit reasoning about the variation or uncertainty present in the real world of the assembly workcell but, instead, leaves the problem of dealing with variation and uncertainty to the on-line execution system. The assembly task is divided into part motions which are realised in the robotic assembly cell by Behavioural Modules, designed and engineered to cope with a certain amount of variation in the workcell. The planner does use some implicit assembly rules, such as putting the parts into the assembly in a vertically downward direction, which help to increase the reliability of the system. Since the earliest system, described here, employs no external sensing (and therefore does not deal with uncertainty), it relies on variation-reducing motion strategies to increase the reliability within the Behavioural Modules in the on-line system.

This section begins by examining the usefulness of the Soma world as a domain for robotic assembly experiments and then continues by presenting an overview of the planning and execution components of the SOMASS system. The reliability of the SOMASS system is then summarised.

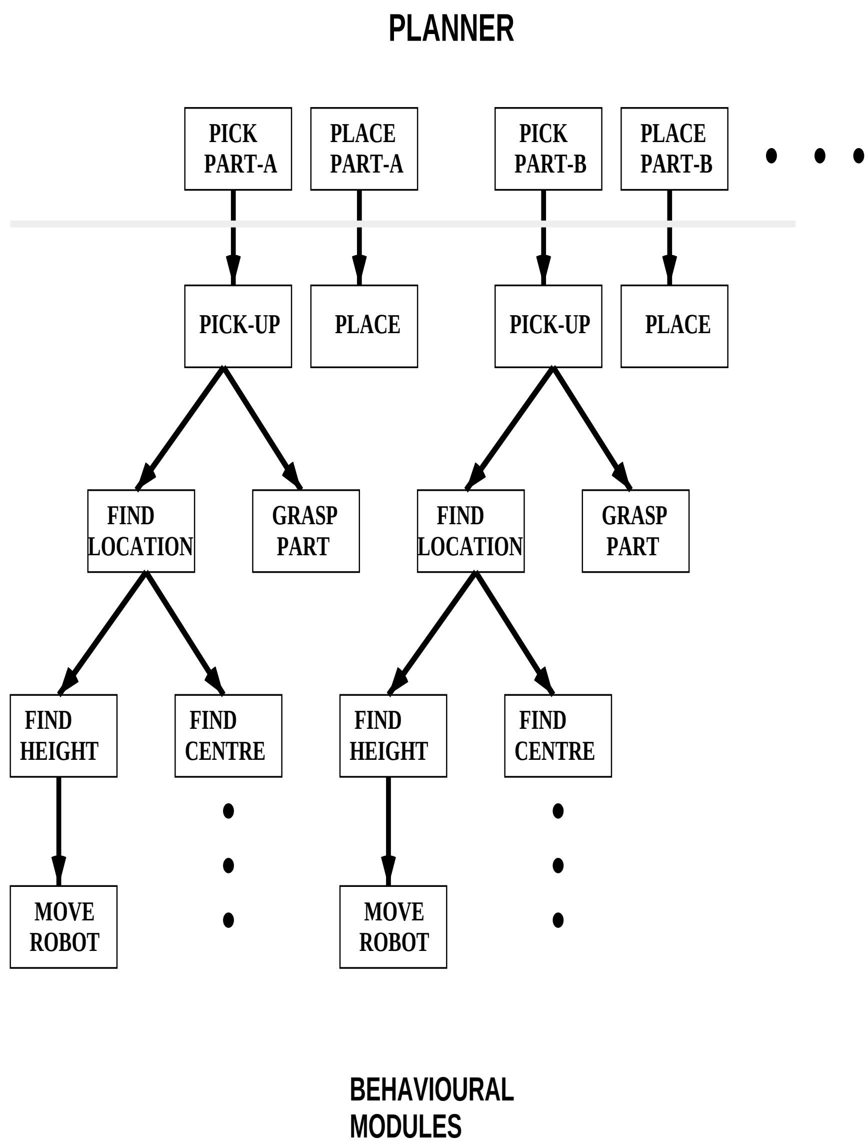


Figure 3–1: A hierarchical decomposition.

ranges from performing the complete task to achieving minor sub-tasks involving single robot motions. The lower level modules tend to focus on the interaction of the robot and the assembly pieces. The Behavioural Modules can be designed by implementing basic low level competences and using these as building blocks to build higher level Behavioural Modules with correspondingly higher competences. Fig 3–1 shows an example of a plan interfaced with a hierarchy of Behavioural Modules.

Emphasis is placed on using information directly from the real world as much as possible, thus moving away from the approach used by more classical systems which rely on a centralised representation of the outside world residing inside the system and updated by sensors. Behavioural Modules are designed, implemented, and tested in the real world, thus ensuring that they are correctly adapted to the chosen environment. As far as is possible, the variation and uncertainty present in the task for which the Behavioural Module is designed, is dealt with within the module by using variation-reducing strategies devised by the designer and by correctly adapting sensors to a particular part of the task. This applies to both the lower level Behavioural Modules which only deal with robot commands, and to higher level Behavioural Modules which can call the lower level ones. The higher level Behavioural Modules may deal with variation and uncertainty reducing strategies over a combination of several different lower level Behavioural Modules.

designed so that sensors could be introduced without significant change to the planner.

3.1.2 Behavioural Modules

The planning system sits on top of the Behaviour-based execution system which is formed from reliable Behavioural Modules which deal with a certain amount of variation and uncertainty in the workcell. These Behavioural Modules are joined together in a hierarchy, with the top level interfaced to the planner. This section looks at the composition of Behavioural Modules and how they can be used to provide a reliable system.

Behavioural Modules can be regarded as modular units which combine hardware and software to reliably perform a task in the real world. The following characteristics are considered to be useful properties of Behavioural Modules in general Behaviour-based systems [Malcolm & Howe, 1990]:

- they handle the variations and uncertainties typical of the task;
- they integrate sensing and action at a low level;
- they are computationally minimal;
- they know (i.e., represent symbolically) as little as possible.

Early work [Smithers & Malcolm, 1989] likened Behavioural Modules to the modular units of VLSI chips. The VLSI design practices produce modular units whose outputs are guaranteed, and they can therefore be easily combined into a hierarchical system. The reliability of the modular units is achieved from control of the uncertainties within the unit.

Built in a hierarchical fashion, each Behavioural Module can be composed of any combination of other Behavioural Modules and individual robot commands in a robot control language. The competence of Behavioural Modules

Traditional planning systems have aimed at the use of symbolic representations of the environment to reason about available actions and the changes in the real world caused by these actions. Such things as perception and motor control can be represented as a set of symbols which can be manipulated by the planning system to produce the final detailed plan. This has led to complex problem solving systems which begin to reason about actions and effects of uncertainty, and deduce what has happened from incomplete knowledge of the world. (See Chapter 2, Section 2.4.2.) This creates systems where the robot motions are carefully detailed by the planner. Error Detection and Recovery systems use the planning system to re-plan the sequence of robot actions if one action has failed to perform its correct function. In Worst Case Analysis systems, the planner bases its actions on detailed models of the world.

The Behaviour-based approach on the other hand leaves the manipulation and sensing to Behavioural Modules which use information from the real world to enable completion of the task, freeing the planner from having to deal with the motions and actions of the robot and the above reasoning about the real world. (See Chapter 2, Section 2.4.3.) The planner, therefore, deals with an ideal world and assumes that the robot will reliably carry out actions such as pick and place. This leaves the planner to deal with problems such as opportunistic scheduling (allowing the changing of the ordering of the assembly depending on the available parts), which involve reasoning about the ordering of the assembly rather than the actual robot motions and sensors used to achieve the assembly.

In the SOMASS system and the experimental system developed for this thesis, a planner which provides a high level description of the task was implemented, leaving the run time details and dealing with uncertainty and variation to be performed at the level of the Behavioural Modules. The planner provides Object Level plans, rather than the motions of the robot to achieve part movement. The translation of the part movement to robot motion takes place in the Behavioural Modules themselves. The SOMASS system was specifically

3.1 System decomposition

Chapter 2 examines the problems facing programmers of robotic assemblies, in particular, that of producing a system which will work reliably in the physical assembly domain. Of the three main methodologies described, Error Detection and Recovery (Section 2.4.1), Worst Case Analysis (Section 2.4.2), and Competent Execution Systems (Section 2.4.3), it is the latter which is used in the thesis work. This section will examine a useful system decomposition which aids the reliability and ease of programming of a Behaviour-based robotic assembly system.

The decomposition of the assembly system falls into two parts: the off-line planning system and the execution system, with the interface provided by a plan. The planner is simplified to aid in the integration of sensing into the on-line execution system, and therefore needs to know little about the physical workcell. The planner can plan in terms of part motions which can be translated into robot motions at run-time. The execution system can be split into modular units, referred to as Behavioural Modules, which are designed, implemented, and tested to reliably perform a particular task or sub-task in the real world.

3.1.1 Planning for a Behaviour-based System

Planning systems are necessary in robotic assembly to impose an ordering on the robot actions or the part motions. Initial constraints determine the ordering of the assembly of the parts, ensuring that one assembly operation does not prevent another from occurring. If a lid is placed on a box before another part is placed inside the box, the lid must be removed again before the latter can take place — an obvious waste of time and effort. Planning can also determine the need to perform sub-assemblies which have to be integrated into the main assembly.

Chapter 3

Behaviour-based Robotic Assembly

This chapter outlines the original ideas behind the development of a Behaviour-based robotic assembly system, SOMASS [Malcolm *et al.*, 1989]. These ideas and the system itself are described here in detail to show how this type of system originally dealt with the problem of providing a reliable robotic assembly in an easily programmed environment.

The first section of this chapter examines the general decomposition of a Behaviour-based robotic assembly system into an off-line planning system and a Behaviour-based execution system. The examination of the execution system concentrates on its decomposition into reliable modular units called Behavioural Modules. The chapter then continues with an overview of the SOMASS system, which is described in detail here as the planning system is used in the experimental work for this thesis and the execution system provided the foundation for the development of the sensory Behavioural Modules. The main emphasis of this chapter is the examination of how the SOMASS system provides a reliable assembly.

ity from a combination of planner enforced strategies, and reliable execution modules, called Behavioural Modules (see Section 3.2.5).

2.5 Methods Summary

The first section of this chapter examined the development of the levels of programming provided for robots. Although these levels have helped the programmer to develop new ways of interfacing with the robot, the underlying problem of dealing with the reliability of the assembly in the real world is not faced. The natural way of programming using these kinds of levels has been seen in the past as updating an internal model of the world with input from sensors, and using this data to determine appropriate actions for the robot. This leads to *ad hoc* methods of integrating sensors, and does not solve the problem of reliable assemblies.

The three main categories of systems designed to deal with variation and uncertainty, as seen above, are Error Detection and Recovery, Worst Case Analysis and Competent Execution Systems. The first of these, Error Detection and Recovery, makes no attempt to avoid failures in the assembly, but concentrates on detecting these failures, diagnosing the cause by consulting some knowledge source, and determining appropriate recovery motions. The main problem associated with this method is determining what has happened at each stage within the context of the assembly. The second method, Worst Case Analysis, relies on being able to model all factors which may cause a system to fail, and determining a successful series of motions under these constraints. This relies on the world being exactly as expected by the system, and often involves extensive computation. The last method, Competent Execution Systems, avoid the problems of planning and modelling the system and rely instead on making appropriate information available when the system needs it. The main research problems associated with this method are finding useful modular boundaries and determining how to integrate high level control to achieve long term goals.

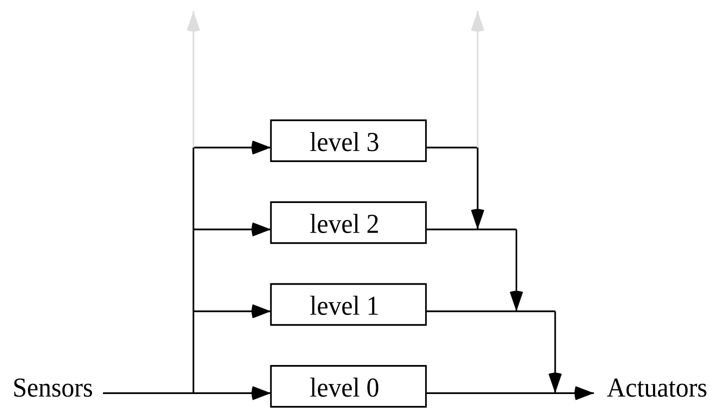


Figure 2–10: Control layers in the Subsumption Architecture [Brooks, 1986].

capability of the robot. Again, each layer was composed of a network of AFSMs. Connell [Connell, 1989] describes a soda can collecting robot designed under the Subsumption Architecture.

So far, this method has proved successful with mobile robots with a small number of limited behaviours. The next stage which is being developed is to test the Subsumption Architecture on more complex robots to achieve more complicated behaviour using the combination of far more modules [Brooks, 1991b] [Brooks, 1991a].

SOMASS System

A brief mention must be made at this point of the Behaviour-based robotic assembly system, SOMASS, developed by Malcolm [Malcolm, 1987] [Malcolm & Smithers, 1990]. This work is described in more detail in Chapter 3, which looks at the SOMASS system as a basis for the experimental work of this thesis.

The SOMASS robotic assembly system is a Behaviour-based system which performs the assembly of Soma shapes, the details of which can be found in Section 3.2.1. The system is divided into a planner which provides details of part motions (see Section 3.2.3), and an execution system which reliably achieves these motions (see Section 3.2.4). This system achieves its reliabil-

in which they live and they manage to act and survive despite the ever changing world around them (see [Brooks, 1986a] for the insect metaphor). Brooks aims to achieve an appropriate level of control for mobile robots.

The Subsumption Architecture consists of layers of task achieving behaviours which together provide the competence of the system. Each behaviour must achieve some task and compound behaviours can be formed by the amalgamation of simple behaviours to achieve more complicated goals. Due to the vertical decomposition of the hierarchical system, multiple parallel routes are available through this system, allowing redundancy and therefore robustness in the task completion.

The Subsumption approach provides an architecture for a complete robot control system for a mobile robot achieving tasks in the real world. A control system is initially built to perform the lowest level task, and is extensively tested. This zeroth level is now never altered and any additional competences to perform new tasks are added by building other layers on top of this (see Fig 2–10). Any new layers can examine data from lower layers and can supply data to these by suppressing the normal data flow within the layers. Each layer is composed of a set of processors, each an Augmented Finite State Machine (AFSM) with no shared global memory. Input signals can be suppressed by signals from the upper layers and output signals can be inhibited. See [Brooks, 1987] for details of the hardware for such a system.

Brooks has gone on to demonstrate the Subsumption Architecture in real robots in real working environments. [Flynn & Brooks, 1988] looks back on the early robots developed at MIT and examines their development from January 1985. The paper examines the development and ideas behind five working robots which have been designed to operate autonomously and robustly in dynamically changing environments. [Brooks, 1989] examines the development of Genghis, a six-legged walking robot which measures about 35cm long and weighs approximately 1 kg. This robot was developed using the incremental steps provided by a subsumption architecture. Basic competences were built, then augmented by additional layers providing an increase of performance and

bottlenecks within the system. Each module can be designed to have low information communication between modules, but each must still await completion of the previous one. This leaves the system vulnerable to the speed of individual modules and to each communication interface between modules. The interfaces must also be set up when the system is initially designed, making it inflexible for future development. Sensor data takes time to filter through the system, and, with this time delay, the external world may have changed and rendered the data invalid (see [Malcolm *et al.*, 1989]). The only way to speed up the process is to enhance individual modules to increase the throughput of information. Updating of internal world models with actual sensor information improves the ability of the system to plan for more than just the immediate actions, but, as already mentioned, if all actions are planned in this way, the time delay may render them inappropriate.

With a vertical decomposition (see Fig 2–9), sensory data can be quickly integrated and processed to produce immediate actions. This means that feedback systems can be easily integrated into the system to guide it to complete the task using information straight from the real world, which leads to more stable systems incorporating less time delay. Each level of the hierarchy can be influenced by its superior, but is independent in performing its own task. The low levels are seen as performing simple and general tasks, with the upper levels looking at more complex and situation specific tasks. Problems associated with these types of system include the integration of high level control to achieve longer term goals, finding good module boundaries where communication is of low complexity, and determining the type of mediation between competing behaviours.

Brooks advocates decomposition of the mobile robot task into task achieving behaviours, based on the above vertical decomposition [Brooks, 1986b].

Brooks has developed a computational model, the Subsumption Architecture, to investigate building control systems for robots with insect level capabilities, in the belief that insects display the reliability and robustness which is really required in mobile robots. Insects are well adapted to the environments

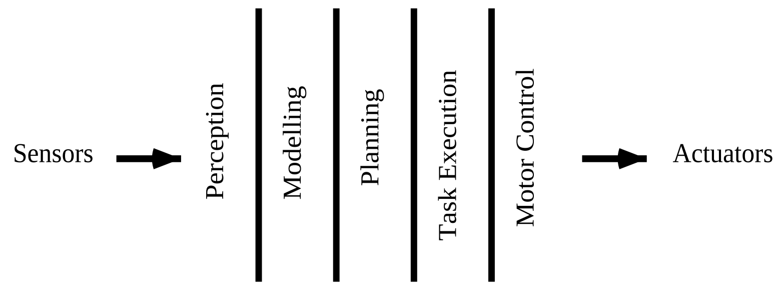


Figure 2–8: Horizontal decomposition of a mobile robot control system into functional modules [Brooks, 1986].

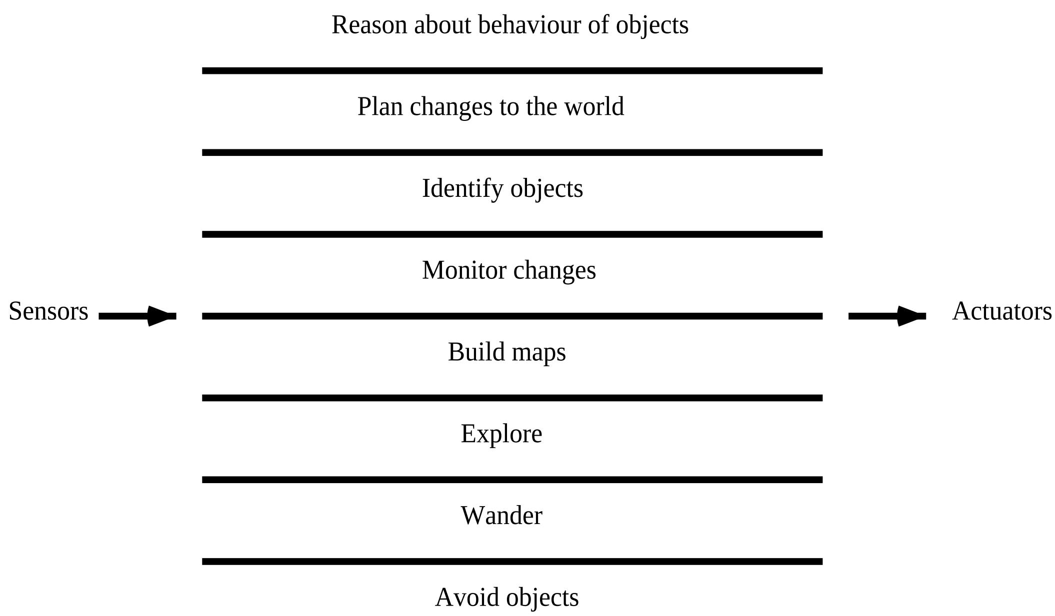


Figure 2–9: Vertical decomposition of a mobile robot control system based on task achieving modules [Brooks, 1986].

programming robots, and the Behaviour-based approach, see modularity as a useful goal and attempt to reduce communication between their modules. It is the way in which the task is decomposed into these modules and thus the composition of the modules which leads to the difference in approach.

Brooks, in [Brooks, 1986a], looks at the difference between the Classical horizontal decomposition and vertical Behaviour-based decomposition of a typical mobile robot task (see Fig 2–8 and Fig 2–9).

In Fig 2–8, the Classical horizontal decomposition, each individual modular unit must complete its task before the rest can begin, causing large information

similar trajectories through these regions, differing only in their magnitude. This leads to an expected route through the sensory space which leads to successful completion of the task. If this trajectory is deviated from when variation in the workcell is encountered, another region of the information space is entered. This region also has associated actions which attempt to converge the present trajectory with the previous expected trajectory. If the trajectory leaves the set of known regions, this is seen as an error, but the data can be examined and a new region incorporated so that in the future, this region will provide an appropriate action.

The difficulty with this system is actually defining the boundaries of the regions in information space. All relevant parameters would have to be examined for a totally defined information space, and it is not obvious what these parameters would be. The actions defined for each region must also be appropriate.

Behaviour-based Programming

A different method for programming robots, the Behaviour-based method, has emerged over the last few years due to a dissatisfaction with what has been termed as the Classical approach to programming robots. The Behaviour-based method attempts to decompose the task into reliable task-achieving units, called behaviours, rather than into the separate information processing functions of the Classical approach. A move is made away from collecting and processing data to form a single representation of the world in which the robot operates, to integrating sensor information at the lowest possible level to direct more responsive behaviour.

The advantages of modularisation are clearly shown by reference to computer programming languages where it is now commonplace to write modular programs, aided by the use of subroutines. An important aspect of modularity is to reduce the complexity between the modules by keeping the communication between modules as low as possible. Both the Classical approach to

servoing and safety. They also deal with their own verification of end-effector position.

Certain assumptions are made concerning the type of environment this system is to work in. These include that variation is typically handled by clever strategies, that the assembly is well constrained in the first place, and that the failures involved are typically the result of certain well understood problems, such as sticking feeding mechanisms. It is also assumed that cheap, fast sensing will be used. The actual sensors used in the workcell are specially designed sentinel binary sensors (a robust finger-like sensor which is activated by the robot knocking its spring-attached top half over), which can be moved round the workcell as appropriate.

Measurement strategies are used to deal with the uncertainty and variation present. Each recognised problem is matched with a specific measurement strategy to deal with it. A strategy of moving a part past a sensor in a specific direction may resolve a problem of a 180 degree ambiguity about the positioning of a part. Variation in feed position can be avoided by being matched with fast and simple correction strategies. If there is no strategy programmed for a particular problem, nothing is done about the problem.

Information Space

Another method of using an execution system to cope with the variation in the environment is proposed by a group at Hull University [Shen *et al.*, 1987] [Selke *et al.*, 1991]. The method involves programming in terms of generic tasks, with each task composed of feeding, transportation, and mating. Each task is characterised in terms of the information required to define success of that phase.

For a particular task, the robot motions are discretised by associated sensor readings into regions in what is referred to as information space. Each discrete region corresponds to an appropriate action and the task can be defined as a trajectory through this information space. Different generic tasks provide

includes a hierarchy consisting of task-decomposition modules, sensory processing modules and world modules. The task-decomposition modules receive information from the sensory processing modules which have extracted sensory data to make decisions at that level. The sensory processing modules are provided with expectations from the world model at the same level. The world model module, in turn, receives input of actions, plans and hypothesis from the task decomposition module. The sensory information can be compared with the world model data to provide feedback to alter the actions. Commands can be represented as vectors and sequences of commands as trajectories.

The complexity of the control of the system is dependent on the number of levels of the control hierarchy, the number of feedback variables, the appropriateness of the H operator and of the sensory processing systems which extract feedback for the H functions.

Supervisory System and Action Modules

A method developed by the Robotics group at Aberystwyth University, designed for the InFACT assembly system, separates sensing and action into different modules, and links the two by a supervisory system [Hardy, 1991] [Loughlin, 1992]. Uncertainty and variation are dealt with by measurement strategies which are only used in situations where it is thought that problems or ambiguities may occur.

The system, which is designed to be easily integrated into existing machine workcells, requires that minimal sensing be used, and machines be easily reconfigurable to allow easy reprogramming for diverse tasks. A supervisory system interacts with action modules, and can also talk directly to a sensor integration box. The sensory integration box collects information from sensors and stores it until it is requested by the supervisor. The sensing and action are deliberately kept separate by the intervention of the supervisory system. The action modules are designed to be internally competent and deal with their own

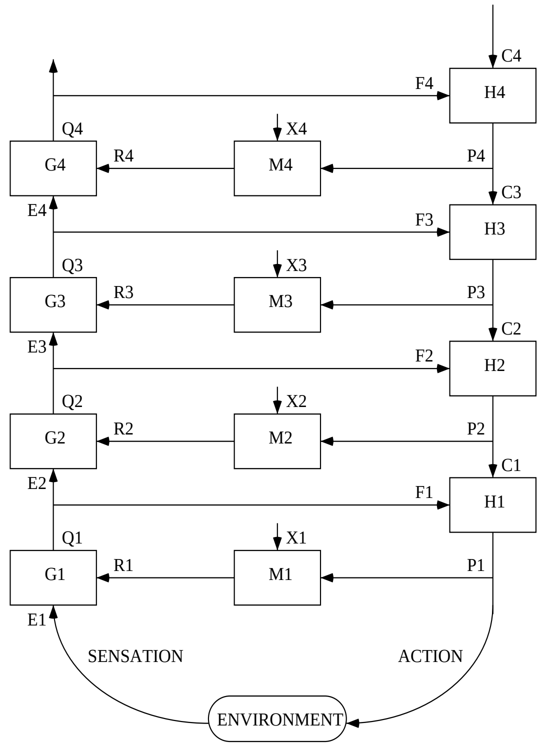


Figure 2–7: A cross-coupled processing-generating hierarchy [Albus, 1981], page 129.

closing a real time control loop at each level of the hierarchy. At the lowest levels of the hierarchy, the sensory feedback is unprocessed and hence fast acting, whereas at higher levels, the feedback provided by the sensory hierarchy has passed through the ascending stages of processing, and thus will be slower acting, giving the higher levels information over longer time intervals. There are several trajectories through the system which will result in successful task completion, but not all result in perfect task performance. If perturbations occur at a low level, unless they overwhelm that level, they can be dealt with quickly. If the system is overwhelmed, other layers of the hierarchy can deal with it by a change of strategy. If the perturbation is so great that none of the levels can deal with it, overall failure will occur. The level at which the system is required to cope with the perturbation gives a good indication of how well it is suited to the task. The further down the hierarchy the perturbations can be dealt with, the better suited the system.

Albus goes on to look at a possible control structure for a robot. This

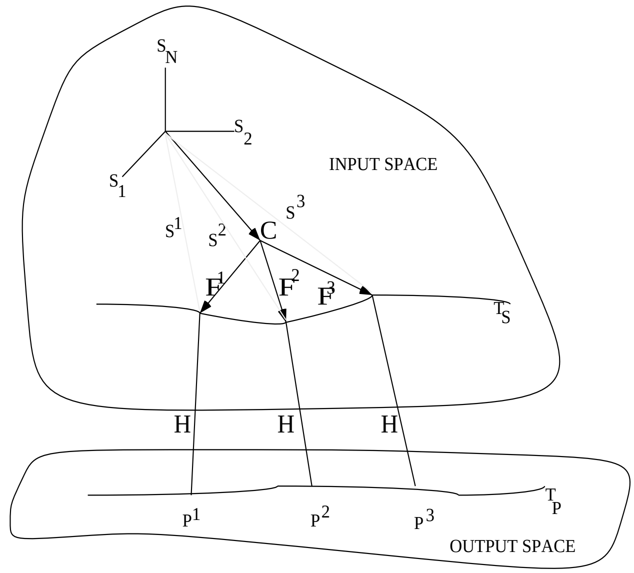


Figure 2–6: Varying Feedback [Albus, 1981], page 115.

tion, in the form of IF/THEN program structures. Sensory data enters at the bottom of the hierarchy and filters up through sensory processing and pattern recognition modules. Relevant information is sent out at appropriate levels.

The H modules in Fig 2–7 decompose input goals C into output subgoals P using feedback F. The M modules recall expected sensory data R which is compared with observed sensory experiences E. The G modules recognise sensory patterns Q and compute feedback errors F. Input to the M modules comes from subgoal information P which indicates what action is being contemplated or executed, as well as from context information X derived from a variety of sources.

The top level input to the hierarchical structure is a goal, which is decomposed into sub-tasks until the lowest level produces observable behaviour. Success or failure of the goal seeking depends on keeping the output within a region of successful performance, despite perturbation and uncertainties in the environment. At each level of the action, or as it is described by Albus, the behaviour hierarchy, feedback is used to amend the current behaviour to keep the system from straying into a failure situation and performs the function of

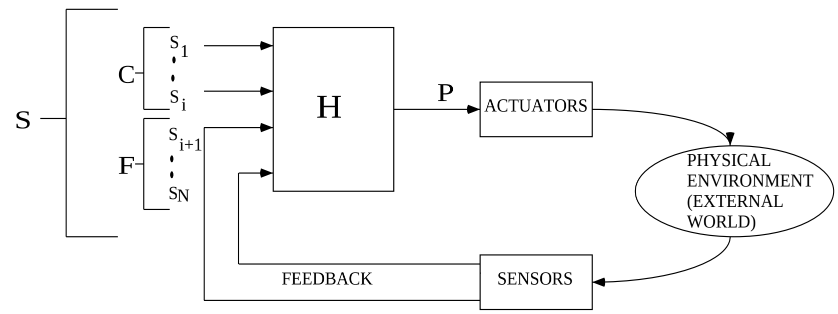


Figure 2–5: A Multivariable servomechanism [Albus, 1981], page 114.

varies over time; the situation is illustrated in Fig 2–6. Albus describes the illustration as follows:

> " A stationary C vector establishes a setpoint, and as time progresses the feedback vector varies from F^1 to F^2 to F^3. The S vector thus traces out a trajectory T_S. The H operator computes an output P for each input S and so produces an output trajectory T_P. The result is that the input command C is decomposed into a sequence of output subcommands P^1, P^2, P^3."

If the command vector, C, stands for a task, a goal, or a plan, then the output string of P^1, P^2, P^3 represents a sequence of sub-tasks, subgoals, or subplans respectively. The system relies heavily on correctly formulated H functions (which may have to take into account speed, stability and the other practical problems which are all embedded in the H functions) over the entire space traversed by the S input.

This type of feedback control including vector notation can be used to produce a behaviour generating hierarchy. A sensory-processing hierarchy is run parallel to the behaviour-generating hierarchy (see Fig 2–7) to integrate both sensory input from the external environment and data recalled from experiences or internal context, to recognise patterns and detect errors. This can then be used to steer the actions along trajectories and provide the feedback which can be used to determine the next action. Memory is used to recall what has happened and predict what will happen depending on a certain ac-

Hierarchical Control

In his book, 'Brains, Behavior and Robotics' [Albus, 1981], Albus attempts to show how the study of robotics can be used as a tool in the study of hierarchical control systems and their relationship to goal-seeking and purposive behaviour. In this, he looks at a hierarchical decomposition composed of goal-achieving modules. This work is presented in detail as the thesis work also uses a similar hierarchical structure, although the modularity is based on different ideas.

Albus uses the notation of vectors and trajectories to describe his ideas. A vector is described as an ordered set, or list, of variables which can specify a state. A trajectory is the path the state vector follows through the vector space as time progresses, producing a sequence of states.

In examining servo mechanisms, the simplest case of using feedback to direct behaviour to achieve a goal, he looks at a hierarchical structure designed to examine achieving goals in an unpredictable environment. Feedback control involves using sensing devices to monitor the state of the physical system, and using this information as part of the next input to the system to moderate the action taken. In effect, the sensory feedback is used to keep the system as close to an ideal path towards the given goal as is possible. Any difference detected between the real world situation and the expected situation results in an action being taken to guide the output in the correct direction to reduce the error.

The vector notation allows expansion of the command, feedback and output used in servomechanisms. The combined input of the command and the feedback from previous cycles is acted on by an operator to produce actuator commands. See Fig 2–5 from Albus. The function, H, operates on all the combinations of the input command, C (a vector consisting of the variables s_1 through s_i) and the feedback, F (a vector consisting of sensory variables s_{i+1} through s_N), to produce the output vector, P (a vector consisting of P_1 through P_L) which controls the actual actuators in the physical environment.

Albus goes on to examine a stationary command vector with feedback which

be represented in this manner in such a way as to provide all the information the system needs to keep running.

When the robot system uses a world model in this way, it becomes important that an accurate world model can be made and the resultant systems tend to become involved in trying to produce an ever increasingly accurate representation of the robot task and work environment. Even so, there is still enough variation in the workcell at run-time to require more than a pre-determined world model to achieve success in the given task. Even small variations in the external world may cause failure of the task, thus making the internal world model an even more inaccurate representation of the real world. Sensors can be added, but the problem arises of determining the correct representation and integration of this new data into the world model. Different tasks require different information from the sensors, which in turn may require different representations. There is also the problem of the time interval between taking readings from sensors, and making decisions based on the new information.

The method of Worst Case Analysis also tends to reject many strategies which may in the real world, correctly perform the assembly. As the bounds put on the system look at the worst scenario which could occur, many useful and efficient strategies are rejected when they may actually work in the real world. This leads to inefficient strategies and the system also considers many situations which will never occur.

2.4.3 Competent Execution Systems

Another form of dealing with variation and uncertainty in the workcell is to design a Competent Execution System. This method plans for a simplified ideal world, and expects the execution system to reliably perform the assembly without resorting to re-planning. The use of reliable modular design is an important factor in this method.

gies which are specifically designed to bring the part into a unique orientation and position in the gripper from a previously unknown initial configuration with substantial initial variation in orientation and position. Other work has concentrated on using 'fences' to manipulate parts in order to achieve a final known orientation. These fences may either be manipulated by a robot with the part on a stationary table, or the part may be moved by conveyor belt through strategically placed stationary fences [Mani *et al.*, 1985].

Peshkin and Sanderson [Peshkin & Sanderson, 1988] have contributed to the above work by placing qualitative bounds on the rate at which a predicted motion occurs to ensure that the methods can produce manipulation strategies which are guaranteed to succeed. This involves calculating the Centre of Rotation (the coordinates of a point which express the two degrees of freedom of the pushed object; its third degree of freedom is constrained by contact with the pusher), which can change dramatically depending on the distribution of the support forces of the object. Thus an attempt is made to find the locus of COR's under all possible support distributions. This provides a basis for planning manipulation of sliding objects either with or without sensors.

Peshkin has since moved on from this type of work to look at programmed compliance [Peshkin, 1990]. Given a nominal motion plan which takes no account of errors, the compliance is then designed in such a way that the plan will succeed. Underlying this strategy is the idea that forces arising during a task often contain enough information to indicate where an assembly has gone wrong. The damping matrix (compliance matrix) contains elements so that the force which characterises every possible error condition maps into a motion that reduces it.

Summary of Worst Case Analysis

In the systems above which use symbolic representation to produce an internal world model from which the robotic system may reason about the current state of the world, an important assumption is made that the world and the task can

In [Donald, 1987], a planner called LIMITED is described which implements EDR in a restricted domain of planar assemblies. Work reported in [Donald, 1988] and [Jennings *et al.*, 1989], expands EDR to cope with multi-step EDR strategies by introducing *Push-Forward Algorithm* and *Failure Mode Analysis*, again implemented in LIMITED. This work is continued in [Donald, 1990] where *Weak EDR Theory* introduces new mathematical tools for studying multi-step strategies.

All of the above strategies involve extensive use of a model of the world and rely on this model taking into account all the relevant parameters which may affect the assembly process. These also, by their very nature, use larger boundaries on the parameters than may be necessary in the actual workcell, thus not allowing some strategies which may be very successful.

Geometric Tolerancing Analysis

Extending Requicha's ideas [Requicha, 1983] on geometric tolerance representation and semantics, Fleming examined the combined effects of tolerancing constraints of assembly parts to determine whether they will fit together satisfactorily [Fleming, 1985a] [Fleming, 1985b]. The geometric tolerances can be defined with tolerance zones (regions of space in which a real feature must lie) and datums (a datum is a point, an infinite line or an infinite plane). In [Fleming, 1987], he shows how a toleranced part can be represented by a network of tolerance zones and datums connected by arcs with associated inequality constraints. The original work was intended for integration into the RAPT programming language to check tolerance during design and off-line programming, but proved to be too computationally expensive for this to be successfully achieved.

The Mechanics of Manipulation

Another category that falls under Worst Case Analysis is that of modelling the mechanics of manipulation. Brost [Brost, 1986] investigated grasp strate-

contact and centre line along the surface normal. A model of friction can thus be made, including angles at which the object will stick or will slide. The model can also be extended to include rotations and moments [Erdmann & Mason, 1988].

Configuration Space and EDR strategies

The previous work concerned with configuration space deals with uncertainty and variation arising from sensing and control errors. Donald [Donald, 1986] proposed a strategy to cope with model errors, that is, errors occurring from uncertainty in the geometric models of the environment and of the robot by parameterisation of possible variations in the environment, such as in length and angle of a chamfer, or width of a hole. It has been discovered that in the presence of model error, there may be cases where such plans do not exist, where the goal may vanish or may no longer be reachable. To deal with this, Donald introduces Error Detection and Recovery (EDR) (a different use of the phrase to that discussed in the previous section) which can only deal with strategies which involve one motion. A plan can be found if the goal is recognisably reachable, but if the goal vanishes, the EDR strategy is designed to allow the plan to signal failure. EDR plans will succeed or fail recognisably with no possibility that the plan will fail without the executor realising it. Model error causes large changes in configuration space, and thus goal failure becomes more prevalent. Donald states,

> "An EDR strategy should attain the goal when it is recognizably reachable, and signal failure when it is not. It should also permit serendipitous achievement of the goal. Furthermore, no motion guaranteed to terminate recognizably in the goal should ever be prematurely terminated as a failure. Finally, no motion should be terminated as a failure while there is any chance that it might serendipitously achieve the goal due to fortuitous sensing and control events."

vector. The range of positions from which motion of a specified velocity along the direction of the unit velocity vector will reach some point of the goal is called the pre-image of the goal. When the pre-image of the goal does not contain the current position of the configuration object, backward chaining can be used to work back until the configuration object is found within the pre-image, using each previous pre-image as the new goal set. From these, a motion strategy consisting of velocity vectors and associated termination predicates can be built up, which can be seen as a sequence of guarded motions.

Variation in the position of the configuration object can also be taken into account by creating a sphere with a certain error radius around the position as observed by the robot. Position variation due to inaccurate knowledge of the initial starting position of objects in the workcell and also the uncertainty in the robot's position sensors are combined to produce an upper error bound. Since this now affects the termination predicates, perhaps leading to premature termination of a motion, all possible positions of the configuration object within this sphere must be within the goal in order to guarantee success. In addition to the variation of the position of the configuration object, there is also uncertainty in the velocity along a motion. The error between the actual and the commanded velocity is given a bound and the resultant path is within a cone centered on the commanded path with apex at the initial position. This gives rise to strong pre-images, which are defined to be the locations for which all motions within the range of velocity uncertainty will reach the goal. To cope with empty pre-images often found when dealing with purely position controlled motions, compliant motions such as sliding can be used along with generalised damping, [Lozano-Pérez *et al.*, 1990a], to produce larger pre-images. Although less sensitive to uncertainty, there still remains the problems of friction and finding the termination predicates.

Erdmann [Erdmann, 1984] looks into the problem of friction in the above work. If it is assumed that the objects are of a single material with equal coefficients of static and sliding friction, the reaction force from contact at a point on a surface will lie within a friction cone with apex at the point of

in the opposite direction to constrain the values for plan parameters to enable the plan to succeed.

Configuration Space

In examining the problem of safe trajectory planning for mechanical manipulators with two links and multiple degrees of freedom, Udupa [Udupa, 1977] presents the basic ideas for *Configuration Space*. A generalised form of these ideas was presented in [Lozano-Pérez & Wesley, 1979] where collision-free paths were planned among polyhedral obstacles. The configuration of a system is said to be any set of parameters which uniquely specify the position of every part of that system, and the configuration space is the space defined by those parameters. It is possible in configuration space to shrink the configuration object down to a reference point and expand configuration obstacles accordingly. The obstacles can be grown more in certain dimensions to encompass most geometrical variation, thus allowing a safety margin for the configuration object to move in this space without collisions and banishing the need for exact calculations for tolerancing error for the obstacles. The problem of planning a safe path for a polyhedral object moving amongst polyhedral objects thus becomes the mathematically easier problem of moving a point through a cluttered environment. Many superficially different path-planning tasks become identical when looked at in configuration space. A more formal explanation of this is presented in [Lozano-Pérez, 1981] [Lozano-Pérez, 1983].

Later work expands the use of configuration space to compliant fine-motion strategies [Lozano-Pérez & Brooks, 1985]. Geometric descriptions of the parts and estimates of sensing and motion errors are used to produce fine-motion strategies which employ a combined strategy of force and position control to guarantee the final configuration from all likely initial conditions.

The basic strategy employed is to look for ranges of positions in configuration space where the configuration object (now reduced to a single point), can reach the goal in a single motion, which can be represented as a unit velocity

used to help robots monitor and calibrate themselves, and control other self-adjusting behaviour. [Johnson & Hill, 1985] uses a Kalman filter to qualify robot move instructions by sensor readings, updating the pre-taught locations with corrected data. These methods give rise to the problems of estimating the statistical parameters required.

Fine-Motion Planning

Early work by Taylor [Taylor, 1976] moves into the area of automatic synthesis of fine-motion planning from *strategy skeletons*. These are partially specified strategies which are completed by decisions made using error estimates found by propagating the effect of errors, variations, and uncertainty through a model of the task. Lozano-Pérez also used strategy skeletons, this time to select motion parameters [Lozano-Pérez, 1976]. Each motion was expressed by the final relationship of the parts during the task. Guarded movements and termination conditions could be computed from the ranges of displacements that achieved this relationship.

Brooks [Brooks, 1982] extended Taylor's approach and presented a plan checker which uses a geometric computer-aided type of database to infer the effects of actions and propagation of errors. This method for checking and modifying robot plans tries to ensure that they will work given mechanical variation in placement, and ranges of tolerances in the construction of the workpieces and their orientation. The result of this is to try to produce a plan which is guaranteed to succeed, by computing the effects of variation and uncertainty symbolically, and, if the effects are small, the plan can be accepted. The computational costs for this system are high. If the effects are significant, most problem areas can be identified and the plan can either be constrained to succeed, or different methods can be used to reduce the variation or uncertainty, and sensory checks included. This method uses explicit models of possible variations. The use of symbolic constraints means that not only can they be used to estimate errors for particular operations, but they can be used

2.4.2 Worst Case Analysis

A different method for tackling the problems of variation and uncertainty in robotic assembly is Worst Case Analysis. This method, unlike Error Detection and Recovery, tries to guarantee that a plan will succeed, rather than letting it fail and recovering from that failed position. Much use is made of both modelling the workcell and mathematical analysis of the task.

Engineering the Workcell

One of the least computationally intensive methods of performing a type of Worst Case Analysis is that of tightly engineering the workcell itself. If the workcell is designed to accept only parts of certain specific tolerances, rejecting completely those outwith these bounds, and fixturing and jigging is used for particular applications to remove other variations, it leaves very little room for error situations to arise. The unfortunate consequence of this is that the method becomes feasible only for large batch products as the development time and cost for changing the workcell for a different product becomes prohibitive. A number of parts are also rejected which may be fitted together, thus causing wastage. Cambridge University [Williams *et al.*, 1986] are working on programmable fixtures to speed up the changeover process and make it more flexible.

Sensory Filtering

The use of control theory has been applied to the problem of uncertainty in robotic sensory data. [Nevins *et al.*, 1984] looks into the advantages which could be achieved by the use of these advanced control techniques in the presence of noisy data and incomplete process models. Kalman filters [Kalman, 1960] [Dean, 1986], have been applied to robotic systems in an attempt to reduce the uncertainty present in the sensory data provided to these systems. [Whitney & Junkel, 1982] show mathematically how stochastic control theory can be

The detection part of the system requires that each action is monitored in an appropriate way, and that the expected state of the system is known. This places a strain on the modelling system to keep track of exactly what is happening in the real world, which is done by updating it as the assembly progresses with information both from the sensors and from expected results of the motions which have been performed. Again, the problem arises of keeping an accurate and up-to-date model which can cope with the actions and consequences of actions in the real world. Each action must have associated current and expected states which, as these are context dependent, mean that each stage of the program must be examined separately.

The problem of context dependency also appears in the diagnostic section where a knowledge base of some sort is consulted to determine what has happened. The sensor readings are thus extracted from the context in which they occurred, and enough information must be transferred with them to determine the actual situation that the particular sensor readings have discovered. It is a difficult problem to determine how much information is needed to correctly identify the appropriate error. For example, if the gripper is empty, is the problem due to gripper failure, or is it because the part has been knocked from the gripper by some unexpected obstacle in the workcell during a motion of the gripper?

The recovery actions rely on the correct diagnosis of the cause of the problem. The knowledge base needs to contain information about the nature of the errors which can occur and, again, the problem arises of information being used out of context where inappropriate actions may be taken.

All the above methods deal with potential error situations, but do not make any attempt to use the software side of the system to reduce the problems of variation or uncertainty. The errors are allowed to occur, then identified, and recovery actions planned. The following section looks at Worst Case Analysis where the systems attempt to provide only plans and robot motions which are guaranteed to work in the real world, given the bounds enforced by the system.

uncertainty and robot motion uncertainty. A patch plan is inserted to link the unexpected configuration to one of the expected configurations [Xiao & Volz, 1989].

Plan Management System

Drabble and Cox [Drabble & Cox, 1985] propose a Plan Management System (PMS) which, as well as planning, carries out execution error monitoring and analysis and can manage plan patching and re-planning. PMS contains three major modules: a traditional planner based on Tate's NONLIN [Tate, 1984]; a system for plan co-ordination, monitoring and error analysis; and a simulation of effectors, sensors and the real world. The main purpose of the research is to have a monitoring system which builds up a string of plan actions which can be replaced by another plan if a problem occurs during execution, a problem being recognised when an action is attempted for which a pre-condition is not met.

One major problem with using planning systems to manoeuvre a manipulator in the real world is that the semantics of the planner are not usually rich enough to deal with spatial relationships, temporal problems, or uncertainty. Many planners create an internal model of the world which is a radically incomplete subset of the real world, containing information which can also be out of date. Facts about the world change as the planner runs due to motions and actions taking place in the real world, thus introducing the possibility of execution errors.

Summary of Error Detection and Recovery

Error Detection and Recovery systems allow errors to occur, then attempt to prevent error situations by diagnosing the failure and planning some new series of motions designed to restore the system. The three sections of these systems, detection of errors, diagnosis of the problem, and recovery actions, each have their own associated problems.

movements are exact imitations of those used by human sheep shearers, it is not feasible to simply attempt to reach the end of a shearing movement by some alternative route. Thus, Trevelyan only allows resumption of the originally interrupted movement rather than trying some alternative method.

Knowledge-based Error Recovery

Researchers at Aberystwyth University have been working to investigate sensory monitoring schemes for error detection, diagnosis of error states from sensory signals, and recovery actions to rectify diagnosed faults [Lee *et al.*, 1983]. AFIRM (Aberystwyth Framework for Industrial Robot Monitoring) has been designed to perform the first of these and to provide information on sensors, sensory states, and controls for the second and third [Lee *et al.*, 1984]. Later work includes an analysis of low level robot actions which, it is claimed, provides a simpler set of errors [Hardy *et al.*, 1989]. In a knowledge base, each atomic robot action is associated with certain general errors. When a sensory mismatch occurs, one or more errors are chosen relating to current action. Possible causes are then sought using a knowledge base. This work is similar to Srinivas in the use of tree structures, but instead of pruning trees from a full failure tree, it attempts to restrict the growth of the tree by applying knowledge as the tree develops.

Replanning for Assembly Tasks

Another method, developed by Xiao and Volz [Xiao & Volz, 1988], plans off-line for a zero error situation and devises simple on-line replanning strategies based upon data available from force, moment, and position sensors to handle errors arising during program run-time. Use of compliant motion is made and restrictions are imposed such that replanning is only performed when unintended contact is detected by force or moment guards between any of the parts involved. Again, extensive use of a world model is made, modelling geometric and physical descriptions of the task environment, along with sensor

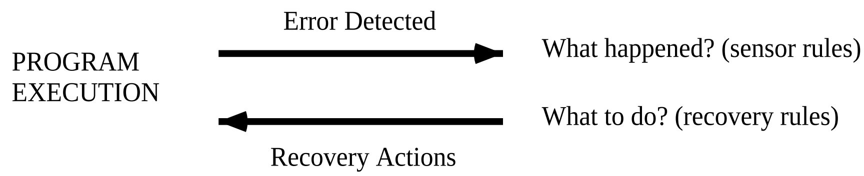


Figure 2–4: Error Detection and Recovery [Gini & Gini, 1983].

A dynamic model of the robot environment is constructed, initially from declarations in the program and data from the sensors. Each robot action has, associated with it, pre- and post-conditions which must match the actual data from sensors. Errors are detected by matching a current model with the expected model for a particular action. In early work, the recovery procedure was only designed to bring the plan back to the point where it was suspended. Later work [Gini & Smith, 1986] expands the Backward Error Analysis and Recovery to include Forward Error Recovery, which consists of determining the difference between the actual state and the desired state, then developing operations to achieve the desired state. Information collected during execution is passed to a recovery process which performs error recovery.

The above two methods rely on being able to correctly execute three different procedures: detect errors, determine their cause, and perform an appropriate recovery action. Each of these has associated problems which will be discussed at the end of this section.

Forward Error Analysis

Another system [Trevelyan *et al.*, 1987] uses a slightly different definition of Forward Error Analysis. Gini [Gini & Smith, 1986] defined Forward Error Analysis as the determination of the difference between the actual state of the robot and the desired state of the robot, then developed a sequence of operations to achieve the desired state. Trevelyan developed a system for sheep shearing in Australia which uses an alternative definition of Forward Error Analysis. Due to the nature of the sheep shearing task, where most of the

he proposed, *Failure Reason Analysis*, attempts to find an explanation for the failure, determine where the problem lies, and work out what can be done about it. A task is seen as a sequence of actions that lead from an initial state to a goal state, with intermediate states defined. A failure results when an action does not produce the next expected state and the problem is then to determine the cause of the failure and to identify a way of progressing from the failure state to the goal state. A failure reason model represents the knowledge about different reasons for failure of an action and what can be done to recover from it. As the task is run, a failure tree is dynamically constructed containing action nodes which are derived from the original plan, and failure nodes. The failure nodes are obtained from an associated action model. It is from a pruned version of this tree that error correction methods and the type of error are sent to a patch planner which integrates them into the initial plan.

The main limitations of this system derive from two sources: the extensive use of plan formation as the basis for constructing robot programs, and from the choice of checking only the preconditions of the actions when the error may be discovered later than it actually appeared which allows the system to go on to perform other actions in the assumption that the previous has been correctly performed.

Backward and Forward Error Analysis

One worker in the field of error recovery for many years is Maria Gini. Moving on from early work in the field of logic programming, Gini proposed a version of software error control using a knowledge base containing rules about the interpretation of sensor data and recovery rules [Gini & Gini, 1983b] [Gini & Gini, 1983a]. A monitoring system is run concurrently with the robot plan which identifies the appearance of any error and consults the knowledge base in order to determine what has happened, and to generate recovery actions. The basic strategy can be seen in Fig 2–4.

Competent Execution Systems — Section 2.4.3 This method allows the on-line system to deal with the variation and uncertainty in the real world, rather than using a method which involves extensive re-planning or modelling.

2.4.1 Error Detection and Recovery

This section deals with the first category for coping with potential error situations, namely Error Detection and Recovery. This method assumes that errors will occur in the assembly process and uses some means to detect these errors, diagnose their cause, and to recover from the error situation either by returning to some previously correct state (sometimes just sweeping all parts from the workcell) or by working forward by some other method to a future correct state. One of the easiest ways of dealing with failure situations by using Error Detection and Recovery is to provide each potential error situation with a fixed error recovery routine. This method uses fixed strategies to deal with events detected by sensors and is thus dependent on the data received and the interpretation of this data.

Many of these systems have concentrated on automating the detection process and recovery planning, and are mainly concerned with monitoring the running of systems with appropriate sensors, and replanning when the current state of the world is different from the expected state. These systems make use of knowledge about the nature of errors to reason about what has happened and how to resolve it.

Failure Reason Analysis

Work with fixed routine error recovery systems led to research in the area of automatic error recovery. Some of the earliest work can be attributed to Srinivas [Srinivas, 1976] and [Srinivas, 1978]. Due to the fact that the world cannot be accurately modelled, Srinivas aimed to design a system which could recover from failures based on knowledge collected about them. The method

2.4 Failure Handling

As the level of programming robots was raised, there still remained the underlying problem of reliability and how to deal with failed assemblies caused by variation and uncertainty in the workcell, the robot, and the assembly components. The programming levels in themselves did not solve the problems so methodologies were built on top of these levels to help to provide reliable systems.

Research into the problem of reliable robotic assembly has led to several strategies for dealing with the problem of variation and uncertainty in robotic assembly systems. These range from the employment of a human overseer to physically restart a failed assembly, to different automatic error recovery methods, and uncertainty and variation reduction strategies. The use of a human overseer is simple and flexible, but is inefficient in terms of human resources.

The methods for dealing with potential failed assemblies fall into three categories:

Error Detection and Recovery — Section 2.4.1 The robotic assembly is monitored for deviations which may result in an error situation. If a deviation is detected, it is identified off-line, and a recovery strategy performed to bring the system back to a point where the assembly can be resumed as normal.

Worst Case Analysis — Section 2.4.2 This method attempts to provide a robotic system which is guaranteed to succeed in the presence of the bounds on the variation and uncertainties presumed in the analysis. This makes extensive use of modelling the world off-line and performing uncertainty analysis on this model.

The integration of ROBMOD also helped the programmer of the system, but did not address the problem of reliability. Fleming [Fleming, 1987] attempted to introduce geometric tolerancing analysis into RAPT, but this proved to be too computationally expensive (this work is also presented in Section 2.4.2 on page 39). Yin [Yin *et al.*, 1984] introduced vision sensing to determine the translational differences between the nominal part position in the workcell and the actual part position. This was done by inserting special RAPT commands in the off-line system to use the camera information at set points. In this way, when these commands were reached, the sensor information could be examined as required. The camera had to be accurately calibrated to fit in with the model of the world already defined by ROBMOD. Although this sensor information proved useful to accurately determine the actual positions of the parts in the workcell, the use of sensors to change the ordering of actions due to real world information, rather than just the destination of the robot arm, could not be easily achieved without bringing much more information off-line from the system. Incorporating this knowledge into the off-line system would prove difficult and specific to a particular workcell set-up.

The above work led to a different line of research — the Behaviour-based approach to robotic assembly. Early ideas behind this line of research can be found in [Smithers *et al.*, 1987] [Smithers & Malcolm, 1989]. This work put forward the idea of programming robotic assembly in terms of task-achieving behaviours. The off-line system plans in terms of part motions, and the assembly is performed on-line by competent execution modules, referred to as Behavioural Modules. Brooks, at MIT, was also developing work in Behaviour-based mobile robotics along a similar theme [Brooks, 1986a] [Brooks, 1986b]. This work on mobile robotics is detailed further in Section 2.4.3 on page 50, and the Behaviour-based programming methodology will be examined in the following chapters. The SOMASS project [Malcolm *et al.*, 1989], a Behaviour-based robotic assembly system developed from the Behaviour-based ideas, will also be fully described in Chapter 3, as this provides the basis for the experimental system developed for the thesis.

In an attempt to make programming easier, the Object Level programming language, RAPT, was developed [Popplestone *et al.*, 1978]. RAPT is a model-based language which allows the assembly task to be described by the relationships ('against', 'fits', 'coplanar', and 'aligned') between the part features ('face', 'edge', 'shaft', and 'hole'). These are entered into the system as follows:

```
<relationship> / <feature1>, <feature2>;

E.g.  against / face of body1, face of body2;
      fits / shaft of body1, hole of body2;
```

From the spatial constraints of the parts and knowing how the part is to be held in the gripper, RAPT is able to infer the position of the gripper to satisfy the constraints by using geometric reasoning. The required motions of the robot could be entered into the system by simply requesting a move using, for example, one of the following commands:

```
move / <body name>;
move / <body name>, parlel, <feature>;
move / <body name>, perpto, <feature>;
move / <body name>, parlel, <feature>, <distance>;
move / <body name>, perpto, <feature>, <distance>;
```

To help the user visualise the above type of relationships, the RAPT language was integrated with a modelling system, ROBMOD [Cameron, 1984] [Cameron & Aylett, 1988] [Ambler *et al.*, 1987]. This allows the objects to be modelled in terms of their surface features which the programmer of the system can then use to define the above RAPT relationships.

RAPT moved away from describing the robot motions in terms of absolute Cartesian destinations, but instead provided an environment where the assembly could be described in terms of how the features of parts fit together.

line developed to produce the Behaviour-based approach currently being followed.

2.3 Robotic Assembly Research at EDAI

This section has been included in the thesis to show how the research work in EDAI into robotic assembly has developed. Beginning with a Manipulator Level system, the level of programming was then raised in the RAPT project, which was programmed at the Object Level. Current research is also focused at this level, with the system architecture developed under the Behaviour-based methodology.

Work in the field of robotic assembly at EDAI began in the early 70's with the 'Freddy' project, which performed the assembly of a toy wooden ship and car [Ambler *et al.*, 1973]. The pieces to be assembled were initially sorted out from a pile, recognised, picked up, re-oriented and placed at their 'assembly position' in their 'assembly orientation'. The actual assembly was a separate program which relied on having the parts placed in exactly the correct positions. Simple vision sensing was used to identify the parts in the first stage of the system, but was not used in the actual assembly. Force sensing was used for two purposes in the assembly section of the system: to determine the success of a spiral search for use in 'hole-fitting', and for some constrained motions where the robot was used to follow a surface by continually bumping into it.

The 'Freddy' assembly system served to highlight several problems which would have to be dealt with if robots were to be used for assembly. Programming Freddy proved to be difficult and tedious where the motions had either to be taught using a teach pendant, or had to be specified in 6D configuration space. This relates to the problems previously described in Section 2.2 with Joint Level and Manipulator Level languages. It also proved to be a difficult task for the programmers to incorporate the sensors into the system.

Summary

The different programming levels, Joint, Manipulator, Object and Task, have been developed over the years to make it easier for robot programs to be developed. As can be seen, this allows robot programming to be performed more quickly and easily for an ideal world, but the problem of performing the assemblies in the real world of the robot workcell still requires that uncertainty and variation are dealt with in some constructive way.

Joint Level programming is too basic to be seriously considered as a useful environment for programming assembly robots. Manipulator Level programs are currently used in actual factory applications, but tend to be used only in tightly constrained environments where the variation and uncertainty allowed in the workcell are kept to a minimum. This is only cost-effective for large batch assemblies. Achieving reliable programs which run repeatedly in the real world without these constraints has yet to be realised. Object Level and Task Level languages also aid the programmer with the ease of visualising the problem and describing the task in familiar terms, but again, there are no guidelines for systematically dealing with the reliability problems.

As has been mentioned in the introduction in Chapter 1, without sensors robots can only execute fixed sequences of motions. Sensor input can be used in two ways: to gather data to change the destination of the robot arm (but still performing the same motion), or to change the sequence of actions of the robot. These uses can overlap where data for the task discovers that an action must be changed. The second use of sensors, to change the sequence of actions, is useful when looking to improve the reliability of the system. The above programming levels, although usually providing a basic interface for sensors, do not provide an obvious systematic approach to their integration.

The next section examines the history of the development of research into programming assembly robots at EDAI. The problems which were encountered by the researchers at each stage are examined, and this shows how the research

this ideal world. An attempt was made to introduce a geometric tolerancing analysis into RAPT [Fleming, 1987], but this proved to be too computationally expensive. Vision sensing was introduced [Yin *et al.*, 1984] to detect the translational variation in the initial position of the parts. The introduction of sensors for this purpose proved successful, but the incorporation of sensors to change the actions of the robot, rather than just its destination, remained a difficult problem. As the sequence of all the positions and motions are determined in an off-line system which knows little about the on-line robot and sensor availability, it is difficult to change the ordering to include alternative strategies dependent on sensor data unless the system is told more about the workcell conditions.

Task Level

In an attempt to raise the level of robot programming still further, another level, Task Level, has been investigated. This level of programming has not been fully achieved, although several systems have been partially developed at this level, for example, AUTOPASS [Lieberman & Wesley, 1977], and more recently, HANDEY [Lozano-Pérez *et al.*, 1987] [Lozano-Pérez *et al.*, 1990b].

The aim of this type of programming is to enable the programmer to specify the parts and the assembly and some goals along the way, such as 'put peg in hole', and the system will do the rest. Thus, the goal of the assembly would be specified, but, to an even greater extent than with the Object Level programming languages, the programmer would have no knowledge of the assembly execution system. All the sensing and robot movement would be hidden from the user. This requires that the off-line system has extensive knowledge of the robot environment as well as that of the parts and what actions can take place in the workcell. This type of system therefore requires the integration of research done into collision free paths, grasping strategies and part mating. Again, as with the Object Level systems, the Task Level systems rely on the hidden Manipulator Level languages to perform the task.

the required spatial relationships between the parts, rather than by describing the robot motions and the end-effector positions as in the lower level systems. This level of abstraction is seen as a more natural way for a human programmer to describe the assembly, and thus makes it easier to program. It is easier to understand what is happening in the assembly when the system is told to align the axis of a peg with the axis of a hole, or place face-1 of one object against face-1 of another, than it is to know what is happening when the robot is instructed to move between two points in the workcell. The actual robot motions to achieve these relationships, usually written in a Manipulator Level language, are hidden from the programmer and are derived by the system.

An example which approaches the ideas behind this kind of system is RAPT, which was developed in EDAI (Artificial Intelligence Department of the University of Edinburgh) [Popplestone *et al.*, 1978]. RAPT is a model-based Object Level programming language, which allows the assembly task to be described in terms of the relationship between the part features. RAPT has a spatial relationship inference system which infers relative locations of parts from a set of spatial relations defined between them [Corner *et al.*, 1983]. These are then post-processed into a lower level robot language to actually run a robot. This system will be described in more detail in Section 2.3 on page 25, which looks at the development of assembly robotics in EDAI.

It is claimed that programming a robot assembly task in an Object Level system can be easily achieved by those not familiar with programming languages. As well as defining the dimensions of the parts and the robot, a good modelling system is required to visualise which features of the objects are being referred to. It is easy to forget which feature of an object was referred to as side-A when it has been rotated to match some previously specified relationship. With a visual modelling system, the current relationships in the assembly system can be easily examined.

Object Level systems rely on the real world of the assembly cell conforming to the geometric model used in the system. Unfortunately, this is not usually the case, as the real world contains variation which does not conform to

times difficult for the programmer to envisage what is actually happening in a long series of robot motions. It is easy to lose track of where the robot should be in relation to the parts, and whether a part has been moved or not.

To aid in the visualisation of the task, many systems, such as GRASP [Derby, 1982] designed at Nottingham University and MCL [Wood & Fugelso, 1983] developed by the McDonnell Aircraft Company of St. Louis, USA, have used graphics tools to simulate the robot environment and workcell. The robot and the parts are modelled in the simulated workcell under which a symbolic language is provided. This is then post-processed into robot executable manipulator level language code. This helps with the planning of the assembly, and collision avoidance problems, but the variation in the real world which is not apparent in the simulated world causes problems when the underlying program is used to run a real robot.

Recently, a group in Germany [Schraft *et al.*, 1992] have linked an assembly robot to a virtual reality head mounted stereo display and data glove system. The robot can be tele-operated by the data glove, and the simulated robot in the virtual reality can also manipulate simulated parts in its environment by using the data glove, but when the operator cannot see the real robot and is relying only on the image in the virtual world, the accuracy needed to perform assembly is not available.

The next section continues the description of the development of robot programming levels with the examination of the Object Level programming languages, which look at describing the task in terms which are more familiar to the programmer.

Object Level

In order to address the problems associated with Manipulator Level programming, a number of attempts have been made to raise the level of abstraction away from the robot motions to deal with the spatial relationships between parts. The assembly program describes the assembly task as a sequence of

guages tend to treat the sensors as input peripherals, and the robot as an output peripheral. This has eventually led to many systems being developed which use a high level detailed model of the world inside the system to coordinate their integration. The workcell and the robot are modelled in this internal world, and sensors are used to update this model with perceived information from the real world. This leads to many problems in keeping this model consistent with the actual world where unexpected events can invalidate the current model, and the time factor between sensor readings and updating the model may further invalidate these. This problem is examined in [Malcolm *et al.*, 1989].

When a robot is programmed off-line, using the combination of position controlled robots and an accurate world model to base the plans of action on, the positions calculated in the model world may not correspond accurately enough to the position in the real world where, for example, a part may be expected to be found. The kinematics of the robot are designed to move the robot to a point in a coordinate frame, and so it becomes important that the positions of the assembly pieces be known in this coordinate system to enable the robot to manipulate them. This problem does not occur at the Joint Level of programming as the positions are taught with the physical robot and parts in place. Robots tend to have the repeatability to move back to taught positions, but may not have sufficient accuracy to move to a calculated position. The Adept robot [Ade1985b], a particularly accurate machine, can return to a taught location with an error of $\pm$ 0.051mm and move to a position taught off-line with an error of $\pm$ 0.127mm. The RTX robot [RTX1987] has a repeatability of $\pm$ 0.5mm in moving to a taught location and its accuracy has been found to be approximately 2mm. Manipulator Level programs have therefore become a combination of teaching appropriate points, and moving the robot relative to these, rather than relying on calculating positions off-line. This reduces the advantage of programming away from the robot and brings some of the programming back on-line.

In programming off-line from the actual workcell and the robot, it is some-

```
.PROGRAM example(pick,drop)    ; subroutine with parameters
;                                pick and drop
  ACCEL 15              ; acceleration 15
  SPEED 50              ; set robot speed to 50
  number = 100          ; number of parts to be assembled

  OPEN                  ; open gripper
  MOVE start            ; move (joint-interpolated) to a starting
                        ; position, where start is given in terms
                        ; of (x, y, z, yaw, pitch, roll)

  FOR i = 1 TO number ; process each part
    MOVES pick          ; move (straight line) to pick up position
    CLOSEI              ; close gripper
    MOVES drop          ; move (straight line) to drop position
    OPENI               ; open gripper
    DEPARTS height      ; departs to a safe height
  END                   ; end of loop

  TYPE ''Assembly completed''  ; end of program
.END
```

Figure 2–3: Example of a VAL-II robot program.

in teaching the robot the assembly task in the workcell. The above method of programming enabled the program to be entered off-line from the robot, allowing the latter to continue with its current assembly task until changeover time. The real robot is only used for fine tuning the system and for making adjustments to the final program, although in reality, as described on page 19, this method of programming still relies heavily on taught positions.

The ability to program the robot away from the workcell, and in terms of Cartesian coordinates in a world coordinate frame, led to the adaptation of current computer languages into robot programming languages. The computer languages retained their flexibility of control structures, including conditional statements, subroutines and loop control, and also added commands specific to the control of robots: to move the robot from one position to another; gripper control; to change the speed and acceleration of the robot; and to allow the integration of sensors. An examination of some of the available languages is presented in [Critchlow, 1985]. VAL [Shimano, 1979], a common robot programming language, was developed using the control structures of the computer language, BASIC, as a model. It did not support floating point numbers or character strings, and could not pass arguments to subroutines. These deficiencies were corrected in a later version of the language, VAL-II [Shimano *et al.*, 1984]. See Fig 2–3 for a small example of a program written in VAL-II. Another robot programming language, PAL [Takase *et al.*, 1981], was developed from Pascal.

Programming robots at the Manipulator Level carries with it associated advantages and disadvantages. The programming skills of the user must be greater than those required for the Joint Level programming, but this higher level allows more flexible control with useful conditional alternative actions depending on either sensor data, or other data available to the system. Programming can be done without using the robot which is safer for the programmer, and allows the robot to be usefully employed while the programming is being undertaken for the next assembly.

With their derivation from computer languages, robot programming lan-

As the programs consist of sequences of positions which must be replayed in order, it is difficult to incorporate any programming control structures which allow alternative robot motions to be used. It is not impossible to incorporate simple conditional statements and loop control structures into a program, but this proves to be a difficult and time-consuming exercise as the programmer must be responsible for keeping a close track of the situation. This is also the case when seeking to incorporate sensors whose readings can determine the next appropriate action to take.

Using this level of programming robots to perform assembly tasks was too unreliable and slow for useful integration into the factory environment. Improvements had to be made in the method of specifying and learning positions, and in the integration of mechanisms such as conditional statements and sensor data control for changing the actions of the robot depending on the actual circumstances in the workcell. The next section examines the development of Manipulator Level robot programming languages, which allows the programming of robots off-line in terms of the position of the end-effector in a defined coordinate frame.

Manipulator Level

The majority of assembly robot programming languages in present use in both factory environments and research establishments are programmed at the Manipulator Level, where the task is described in terms of the position of the robot's end-effector in some known coordinate frame, usually relative to the base of the robot. (See Fig 2–2.) The robot joint values are calculated from the coordinates of the position of the end-effector, given in the form of x, y, and z coordinates, plus orientation values for the gripper. These Cartesian coordinates are accepted as input and converted using kinematic transformations into joint angles at either compile or run time. See [Craig, 1989] for more details.

This method of programming was inspired by the problems encountered in the Joint Level programming systems. Too much time was being spent

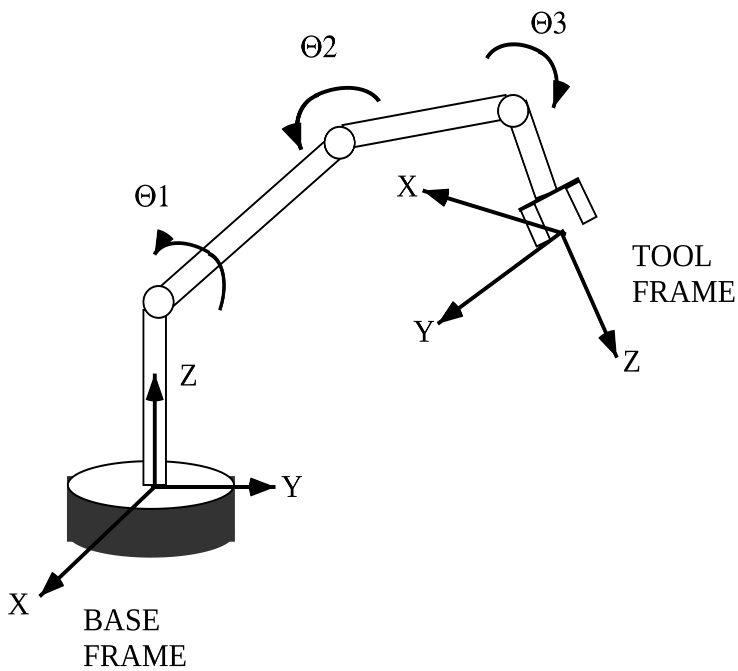


Figure 2–2: Relationship between the robot end-effector and joint angles [Craig, 1989].

could also be traversed in the reverse direction. Programs became a sequence of positions, which were usually taught to the robot system by physically moving the robot joints to the required positions, and storing the sequence.

The advantages of this kind of programming are few. It is easy to know where the robot is going to move to, as the points have been stored by actually placing the robot in the physical position, but actually teaching the robot these positions requires that the programmer work in close proximity to the robot, which can prove dangerous. It can also be extremely difficult to judge the alignment of axes by line of sight alone, and viewing from different angles becomes necessary, again placing the programmer within the workcell. To develop new programs to assemble different parts, both the robot and prototype examples of the parts must be available, thus keeping the robot from being usefully employed in performing actual assemblies. Although computational costs are low, small changes in the assembly task can require extensive changes in the robot program.

quickly and reliably. The following sections look at the four main programming levels, described in [Malcolm & Fothergill, 1986]. These are:

Joint Level: Specifies the end-effector position in terms of the joint angles or joint displacement of the robot which must be achieved to place the end-effector in that position.

Manipulator Level: Specifies the end-effector position in terms of Cartesian coordinates in a known coordinate frame.

Object Level: Specifies the required spatial relationships between the features of the parts to be assembled.

Task Level: Uses a high level specification of the assembly task, such as, 'place peg in hole'.

The first two levels are concerned with moving the end-effector to a known position in the workcell, and the last two levels address the problem of describing part motions. These levels will now be examined in more detail.

Joint Level

At this level of programming assembly robots, the position of the end-effector is given in terms of the individual joint angles which must be reached to achieve that position. See Fig 2–2 (from [Craig, 1989]). At the simplest level, this involves providing a control feedback loop for the servo-mechanisms which control the joints of the arm, to compare their goal positions with the actual positions and to work to reduce the difference by moving in the right direction.

Early robot movement control techniques led to erratic robot arm trajectories as each joint was activated at the same time whatever the distance it had to travel to its final destination. This method was refined to allow the links to begin and end their movement simultaneously by running the motors at different rates. This joint-interpolated motion provided much smoother paths which

ation. An error situation occurs if, were the robot to continue normally with the task, the task would not be correctly completed. A distinction is made here between an error situation and a situation from which the robot, with some extra actions, continues the assembly to a successful conclusion. An error situation requires some external agent, such as a human, to intervene to restart the system.

In a sensorless robotic assembly system where no measurements can take place, the system will not be creating uncertainty, except in the internal sensors of the robot, but will still have to deal with any variation which occurs within the workcell. This is usually done by performing clever motion strategies, or by using jigging and fixturing. If the system employs sensors, not only does it have to cope with variation, but has to take into account any uncertainty these introduce. The close linking of sensors with the task in hand helps to alleviate this problem, as inappropriate matching of sensor with task and the environment may lead to a build up in uncertainty. Even if the right kind of sensing is being used for a particular task, if the useful information range of the sensor coincides with the level of noise in the environment or the noise generated by the sensor itself, the problem of uncertainty build up may occur.

2.2 Historical Perspective

In order to more fully understand the nature of the problem facing programmers of robotic assembly systems, a historical look must be taken at the development of robot programming languages and how these have affected the way in which systems are produced. Through the years, in an attempt to facilitate the adaptation of robot programs to different assemblies in the factory environment, the level at which robots are programmed has changed. To achieve the integration of robots in factories, it must be possible for someone, not necessarily familiar with advanced programming techniques, to program the robots

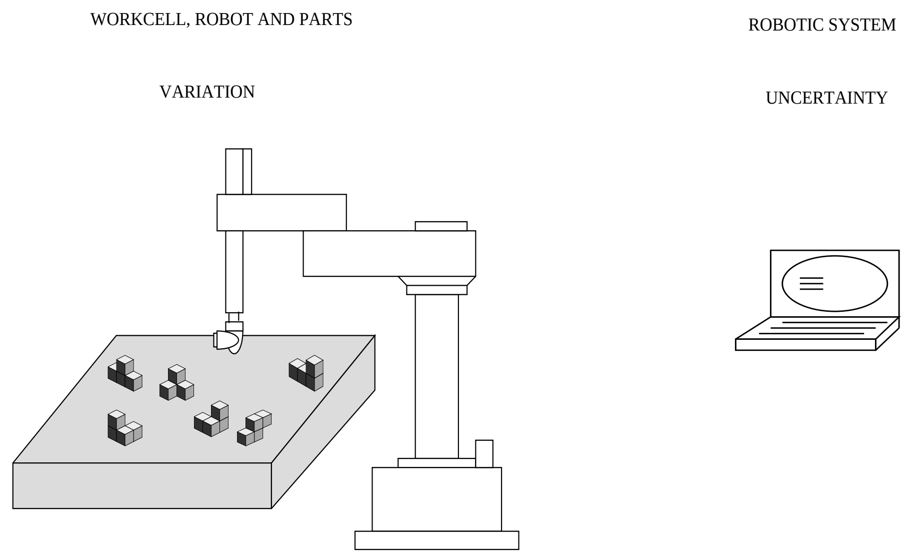


Figure 2–1: Variation and Uncertainty.

When the sensors are used to measure things in the world, this does not actually reduce the variation in the workcell which is constant until some outside agent alters this fact (as in moving the part). Variation can be reduced or changed by physically changing the location or dimensions of the part and various strategies can be employed to use the robot to do this. The robot system can be uncertain about the variation in the world because it tried to measure the amount of variation. The diagram in Fig 2–1 shows the relationship between variation and uncertainty within a robotic assembly system. The variation is present in the workcell and the uncertainty is a feature of the robot system.

- **Error Situation**

  If the variation actually present in the system is greater than it has been designed to cope with, or the uncertainty in measurements taken from sensors proves to be too much for the system to deal with, or a combination of the two, the robot may fail to correctly complete the assembly. If the robotic system fails to perform its allotted task or sub-task, and no built in recovery action is taken, this failure results in an error situ-

Depending on the way the parts are supplied to the workcell, there can be a large variation in the position in which they are presented to the robot. The feeding mechanisms can vary from presenting the parts in tightly engineered pallets which, although expensive, significantly reduce such variation, to the parts being randomly positioned in a large bin. The relative costs of these methods contribute to the cost-effectiveness of the robot workcell. If the parts are accurately placed, the feeding mechanism and tight tolerancing become expensive, and if they are randomly placed, the cost goes into dealing with the large variation in the way the parts are presented.

In the real world, variations occur not only in the workcell and the parts to be assembled, but also in robot itself. The robot kinematic model may be an inaccurate representation of the real robot, and different behaviour under load, such as bending, may further invalidate this model. Temperature variation, structural faults, and joint wear may all contribute to this inaccuracy and these may vary from run to run even when performing a seemingly identical task.

Uncertainty, on the other hand, as distinct from variation, is only present when using the measured values from sensor readings. If, for example, a sensor has the capability of measuring the position of a part to within $\pm$ 1mm, that sensor can be used to locate that part to within $\pm$ 1mm whatever the variation of the part in the workcell, giving the system the uncertainty of $\pm$ 1mm. Much work has been undertaken in reducing the value of the uncertainty associated with sensors, and some of this work is examined in Section 2.4.2, page 34 where Kalman filters are discussed. Sensors can be used to locate a part more accurately, that is, lessen the effect of variation, but in doing so, account must be taken of the uncertainty which the system is then subject to. Using the definition of uncertainty above, the use of sensors actually introduces uncertainty into the system (but can be used to reduce the effects of variation on the system).

systems and methods. The chapter concludes with a summary of the methods described and the problems which still remain.

2.1 Background to Variation and Uncertainty

In the robotic assembly domain, there are two main sources of problems which may lead to assembly failures. These are:

1. Variation in part shape and dimensions, and in their positioning within the workcell.

2. Uncertainty introduced by sensor measurements.

For the purpose of this thesis, the terms *variation* and *uncertainty* are specifically treated as independent problems which can be dealt with separately but which can both contribute to problems and failures. In particular, the term uncertainty is used differently from the normal usage found in the literature where it is often bound together with variation. These terms are fully defined below. The term *error situation* is also used to specifically describe failed assemblies where operator intervention must take place, as opposed to situations where a problem is detected and recovered from automatically by the system.

- **Variation and Uncertainty**

  When an assembly system is run, the parts supplied to be assembled will vary in dimension. Variation can be seen as the difference between the nominal position or part size, and the actual. Part size may range within certain allowable variations, known as tolerances, or may, on rare occasions, exceed these bounds. Most current assembly systems rely on being supplied with parts which are within predefined tolerance bounds to successfully complete the assembly. It is comparatively rare, again depending on the actual assembly pieces, to be provided with parts which have missing or additional features.

Chapter 2

Related Work

The purpose of this chapter is to examine the problems which face programmers of robotic assembly systems, in particular, the problem of failed assemblies due to *uncertainty* and to *variation* in the robot, workcell, and parts (see Section 2.1 for the specific meanings of these italicised terms which are used throughout this thesis). To do this, a detailed look into the history of robot programming methods and current research is required.

The first section of this chapter introduces the terms uncertainty, variation and error situations. Programming languages to control robots have been developed through the years in an attempt to provide a level of programming which allows the programmer to quickly and easily develop reliable assembly programs. The second section examines these programming levels and comments on their associated advantages and disadvantages. The robotic assembly research work in the Artificial Intelligence Department of the University of Edinburgh (which will be referred to as EDAI), which provides the foundation for the work of this thesis, is then examined, from the early 'Freddy' system to the current Behaviour-based line of research. The chapter continues by describing the different ways in which researchers have tackled the main research topic of this thesis: increasing the reliability of robotic assembly. This is divided into three parts: Error Detection and Recovery, Worst Case Analysis, and Competent Execution Systems, with each part sub-divided to deal with particular

Chapter 5 This chapter describes the experimental system which was developed to investigate the ideas of Chapter 4.

Chapter 6 The results of test runs of the above experimental system are presented.

Chapter 7 A summary of the thesis work is provided and suggestions for possible extensions of this work are detailed.

This type of structuring can be used to increase the reliability of an assembly system. Using monitored Behavioural Modules which are guaranteed to provide one of several known Exit States, possible error situations can be identified and alternative methods for handling failures devised.

The above system architecture can be used to contain the variation and uncertainty in the on-line system, thus freeing the planner from any reasoning about these matters. The on-line system therefore has the freedom to deal with the interpretation of the part motions into robot motions depending on what strategies and sensors are available to it. The robot motions are not dictated by the planner, and this also allows easy implementation of improved strategies, and the integration of alternative sets of actions if required.

An experimental system was designed and implemented for the thesis which developed Behavioural Modules in the above type of architecture to provide a working system which was built and tested in the real world.

1.4 Outline

The structure of the rest of the thesis is as follows:

Chapter 2 This chapter examines the robot programming language levels to show how these have been used to attempt to produce easily programmed systems. A survey of the methods which have been used to try to achieve the goal of producing reliable robotic assembly systems is also presented.

Chapter 3 An introduction to the Behaviour-based approach to robotic assembly is presented along with a detailed overview of the SOMASS robotic assembly system.

Chapter 4 This chapter introduces the ideas of the thesis and defines the terms which are used to describe the work.

1.3 Reliable linking of Behavioural Modules

This thesis examines a method of using appropriate information available to the on-line system along with information returned from lower level Behavioural Modules in the hierarchy to reliably link the Behavioural Modules and provide alternatives on failure of a module. This uses the concepts of *Exit States*, the *Ideal Execution Path* and *Alternative Execution Paths.*

The Exit State of a Behavioural Module informs the system what has occurred within that Behavioural Module, and reflects the control paths available inside it. The control paths can vary within a Behavioural Module depending on the returned Exit State from lower level Behavioural Modules in the hierarchy and from deviations determined by local information (usually specific data from the plan in combination with on-line knowledge of the physical workcell). The Exit State gives an idea of the competence of the Behavioural Module and what it can cope with, although, as Behavioural Modules can be general purpose, the context in which it is called determines whether it has actually performed its required task. In one call of a Behavioural Module, the Exit State 'A' may mean successful completion of the task, but in another, this Exit State may mean that the Behavioural Module has failed in its assigned task. The Exit State can also be qualified by other parameters.

The Ideal Execution Path and Alternative Execution Paths define control paths through the hierarchy of the Behavioural Modules. The Ideal Execution Path describes the best route which can be taken through the Behavioural Modules in the on-line system (determined by the system designer). Even though Behavioural Modules are designed to perform reliably, there are still times when they can fail. Alternative Execution Paths can be used if particular Behavioural Modules on the Ideal Execution Path fail to perform their given task correctly during the assembly, to attempt to bring the control of the assembly back to the Ideal Execution Path.

process to enable useful incorporation of sensory data at appropriate points. The main research problems associated with this method are finding useful modular boundaries, determining the modular architecture of the system, and deciding how to integrate high level control to achieve long term goals.

The work in this thesis examines the problem of variation and uncertainty in the robotic assembly workcell and looks at how it can be dealt with in the framework of a Behaviour-based robotic assembly system.

1.2 Behaviour-based Robotic Assembly

The Behaviour-based approach to programming assembly robots aims to provide a decomposition of the assembly task into easily programmed modules, referred to as Behavioural Modules, which reliably perform their task in the real world. Structured in a hierarchical fashion, with the top level determined by the interface to a simplified planner, the Behavioural Modules, which combine hardware and software to perform their task, can be individually designed, implemented and tested to help determine their suitability and reliability for a particular task.

The planner can plan in terms of part motions, which are easier for programmers to identify with than robot motions, and these part motions can be translated into appropriate robot motions in the run-time system. This improves the flexibility of the system by allowing the run-time system the freedom to implement the conversion as required, and change it when necessary. The assembly is now the explicit goal of the plan, rather than the side-effect of robot motions.

fall into the categories of Error Detection and Recovery, Worst Case Analysis, and Competent Execution Systems.

Error Detection and Recovery systems approach the problem of providing reliable systems by allowing errors to occur during the running of the assembly and attempting to perform actions to recover the system. The assembly system is run until a failure is detected, a diagnosis is made of the cause of the problem, and recovery actions are determined and performed. Much research has gone into automating the above processes, with concentration focusing on introducing sensing to monitor the workcell in order to detect the failures, and setting up knowledge bases both to determine what has occurred and to decide upon the best course of action. To enable the detection of errors, these systems compare expected sensor values (determined from global world models) and actual sensor values. This requires that the global world models be kept up-to-date with appropriate context-dependent information. It is not an easy problem to determine what information from the running system should be transferred to the knowledge base to determine the cause of the error, and this removal of the contextual information from the actual on-line system means that incorrect diagnosis of particular instances of problems can occur.

Worst Case Analysis systems concentrate on producing plans which will never fail, given the constraints imposed on the system by the designer. Much use is made both of modelling the workcell and mathematically analysing the task, leading to computationally expensive systems. By the very nature of this work, it looks at dealing with the worst possible scenario which could be faced by the system, and attempts to find a plan which will work in these tight constraints. This precludes many strategies which will actually work in the real workcell, and often rejects many which are useful and efficient.

Competent Execution Systems (one type of which, the Behaviour-based approach, is used as the foundation of the thesis work) plan for a simplified ideal world and allow the execution system to reliably perform the assembly without recourse to re-planning at any point. These systems rely on making information available when the system needs it and modularising the assembly

Robots are already usefully employed in areas which, for humans, would be dangerous, such as undersea work or toxic environments. Unpleasant and repetitive tasks such as spray painting and arc welding are also performed by robots. Among the advantages of the use of robots is that they are tireless workers, performing consistent work over extended periods of time. It is the problem of providing a reliable and easily programmed robotic assembly system which is addressed in this thesis.

1.1 The Reliability of Robotic Assembly

A major problem with the use of robots to perform assembly tasks is the presence of variation and uncertainty in the workcell. Variation in part dimensions and positioning, and uncertainty introduced when using values measured from sensors, can combine to produce an error situation which requires human intervention to restart the system after an unsuccessful assembly. This has led to the production of tightly engineered robotic systems which, like specialised hard automation systems, are specific to a particular application and lose the generality and flexibility originally promised by the introduction of robots to assembly.

Without sensors, robots are limited to performing pre-programmed motions and cannot make decisions based on information gathered from the real assembly world. In fact, all they can do is slavishly perform a series of motions, regardless of the side effect of performing an assembly. Unfortunately, the real world is not perfect, and, although complicated robot motions can be performed to try to ensure that the task is successfully completed, it is often useful to integrate sensors into the system to allow decision making dependent on this information and to check whether a task has been correctly performed.

Research on the subject of producing a reliable robotic assembly system has focused in three main areas which are developed further in Chapter 2. These

the other hand, fatigue through a working day makes it difficult for humans to maintain a constant performance, leading to products of varying standards and a problem with quality control. Certain tasks must be performed in potentially dangerous and unpleasant environments where bulky protective clothing must be worn, thus limiting the efficiency of the worker and reducing the acceptability of the working conditions.

Hard automation assembly systems with dedicated machine tools provide repeatable results which lead to a good level of quality control. Dedicated systems deal with the variation and uncertainty in the workcell by the use of a large number of special jigs and fixtures. These serve to reduce significantly the possibility of failure of the system by tightly engineering the environment. This leads to systems which are expensive to change between different assembly products as the workcell is specifically designed for only one. Even a small change in the size of the parts may require extensive changes in the fixturing and programming. The parts must be made to within tight tolerances or the system will fail, adding to the problems of quality control, and the feeding mechanisms must present the parts accurately, adding considerably to the cost of the system. Sensory systems are very rarely used for anything more than detecting the presence or absence of parts at various stages of the assembly, relying instead on accurately setting up the non-sensory components of the system to enable the successful performance of the assembly. In industrial assembly, where the primary concerns are reliability and cost-effectiveness, this method is only cost-effective when applied to large batch runs of a particular product.

Unfortunately, only a minority of goods which can be assembled have the advantage of large volume, the rest requiring small batch runs before changeover to a different product. In some cases, it would also prove useful to interleave different assemblies depending on the demand for particular products and the availability of parts. The current hard automation is unable to deal with this type of flexibility. It is in this area that it is hoped that robots can be introduced to fulfill the potential of their apparent flexibility.

Chapter 1

Introduction

The reality of dexterous, walking, talking, and thinking robots of popular Science Fiction tales is still far from being realised in the present generation of robots and is likely to remain so for many years to come. The robots of today still have primitive motor control and limited sensory capabilities despite the advances made in computer technology, engineering, and sensory equipment. Even robots designed specifically to perform assembly tasks lack the dexterity and sensory capabilities of a human hand. Mobile robots encounter problems attempting to duplicate the degree of competence displayed by simple insects and the complex capabilities of a human are still a long way beyond their current skills. Despite the above lack of ability, robots can still perform useful functions and, as their capabilities improve, this range will increase.

Many industrial assemblies are performed by humans, who are readily taught new applications and are quick to adapt to different situations. The advanced sensory skills possessed by humans, in particular, touch, force, and vision, allow skillful handling of assembly pieces, including compliant motions which are particularly difficult to control in most automated systems. With the application of intellectual reasoning and basic common sense, directed experimentation or previous knowledge of possible problems can lead to a quick diagnosis of the fault if assembly pieces cannot be correctly fitted together. The part, or parts, at fault can be discarded, or recovery actions can be performed. Human assembly workers can easily adapt to changes of assembly product. On

List of Tables

List of Figures

Table of Contents

Declaration

I declare that this thesis has been composed by myself and that the work described in it is my own.

Myra S. Wilson

Acknowledgements

I would like to thank the following for their help during the preparation of the thesis: Dr Tim Smithers for his guidance, time, and constant support, Chris Malcolm for developing the SOMASS project which provided the foundation for this work and for his supervision in the later stages of the work, Howard Hughes for proof reading and for his comments throughout the years of sharing an office, and Jonathan Salmon for reading early drafts and for some proof reading. Thanks also to the technical staff for help with the practical side of robotics and to SERC for funding.

Thanks also to the following members of the ARG robotics discussion group for both technical and moral support: Alistair Conkie, Mitch Harris, Lykourgos Petropoulakis, Piak Chongstitvatana, Peter Balch, Graham Deacon, and Jim Donnett.

In addition, I would also like to thank the following past and present members of the department who have helped to make my time here enjoyable, either as dining companions, games players, or sports partners (or any combination of the three): in particular, Mike Cameron-Jones, Stephen Cranefield, Diana Bental, Barbara Webb, and also all the others who are too numerous to mention here.

I would like to take this chance to thank the "Mechanical Engineers" who welcomed me into their badminton team and also the KBG for many an energetic Monday evening.

I would also like to thank my flatmate, Marianne MacLeod, for putting up with piles of assorted papers littering the flat for the last few months, and for refraining from dumping them with the newspapers and bottles to be recycled!

Last, but most certainly not least, I would specially like to thank my parents and my sister for their constant support (and chocolate praline!) throughout the course of the thesis.

Abstract

The research in this thesis looks at improving the reliability of robotic assembly while still retaining the flexibility to change the system to cope with different assemblies. The lack of a truly flexible robotic assembly system presents a problem which current systems have yet to overcome. An experimental system has been designed and implemented to demonstrate the ideas presented in this work. Runs of this system have also been performed to test and assess the scheme which has been developed.

The Behaviour-based SOMASS system looks at decomposing the task into modular units, called Behavioural Modules, which reliably perform the assembly task by using variation reducing strategies. The thesis work looks at expanding this framework to produce a system which relaxes the constraints of complete reliability within a Behavioural Module by embedding these in a reliable system architecture. This means that Behavioural Modules do not have to guarantee to successfully perform their given task but instead can perform it adequately, with occasional failures dealt with by the appropriate introduction of alternative actions.

To do this, the concepts of *Exit States*, the *Ideal Execution Path*, and *Alternative Execution Paths* have been described. The Exit State of a Behavioural Module gives an indication of the control path which has actually been taken during its execution. This information, along with appropriate information available to the execution system (such as sensor and planner data), allows the Ideal Execution Path and Alternative Execution Paths to be defined. These show, respectively, the best control path through the system (as determined by the system designer) and alternative control routes which can be taken when necessary.

Achieving Reliability using Behavioural Modules in a Robotic Assembly System

Myra Scott Wilson

Ph.D.

University of Edinburgh

1992