



GeRT



Generalising Robot Manipulation Tasks

www.gert-project.eu

Christoph Borst (DLR – Robotics and Mechatronics Center)

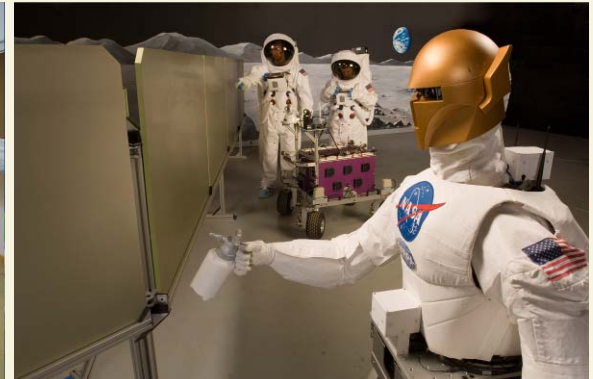
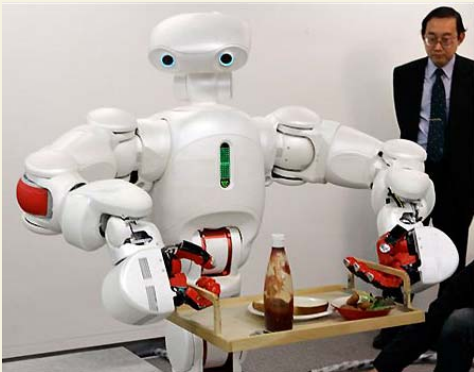
23.2.2012

Consortium

Beneficiary Number *	Beneficiary name	Short name	Country	Main research objectives
1 (coordinator)	Deutsches Zentrum für Luft- und Raumfahrt	DLR	Germany	<ul style="list-style-type: none">• Visual machine perception• Robot control• Integration and demonstration platform
2	University of Birmingham	UB	United Kingdom	<ul style="list-style-type: none">• Learning of planning operators / hybrid planning• Optimising Grasping Strategies using POMDP solvers
3	Oerebro Universitet	ORU	Sweden	<ul style="list-style-type: none">• Symbolic robot plans / hybrid planning• Task level planning• Geometric reasoning
4	TU Darmstadt	TUD	Germany	<ul style="list-style-type: none">• Adapting grasps and grasping primitives to new objects using statistical reinforcement learning

Motivation

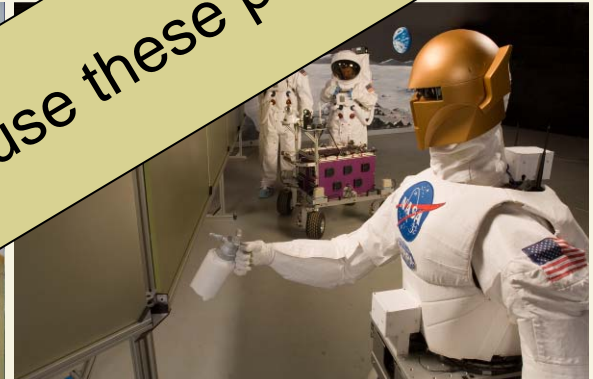
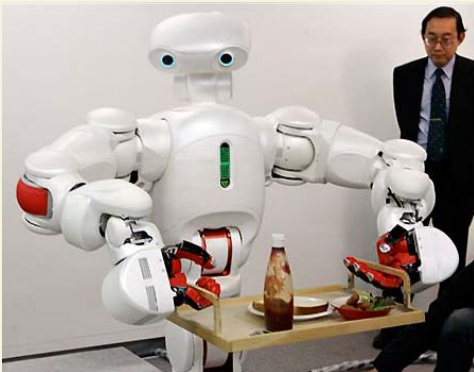
- Robots already perform nice manipulation tasks / demos
- Special programs for each **robot**, **task** and **object**
- Robust programs need expertise and time!



Motivation

- Robots already perform nice manipulation tasks / demos
- Special programs for each **robot**, **task** and **object**
- Robust programs need expertise and time!

Can we reuse these programs?



State of the art

- Manipulation options are currently restricted to:
 - Small set of previously known objects
 - Object recognition and localisation
 - Geometry of objects is known in advance
 - Minor changes in the scene arrangement
 - Motion planning (goal configuration reachable)
 - Offline grasp planning (take a configuration from database)
 - Simple Objects and simple tasks
 - Robustness of operation is far from suitable for daily use

Motivation

In GeRT we aim to generalise manipulation task with:

- the same task constraints
- different object geometries
- different object positions



Motivation

Goal:

Enable a robot to autonomously generalise its manipulation skills from a set of known objects to previously un-manipulated objects in order to achieve an everyday manipulation task.

Approach:

- A set of demonstration programs achieving the same abstract task with different objects and varying scene arrangements is coded by hand and executed on the robotic system.
- These examples form the base for generalising the planning operators and for learning pre- and post conditions of operations.

Key aspects

Planning

- Create programs for new task variations

Perception

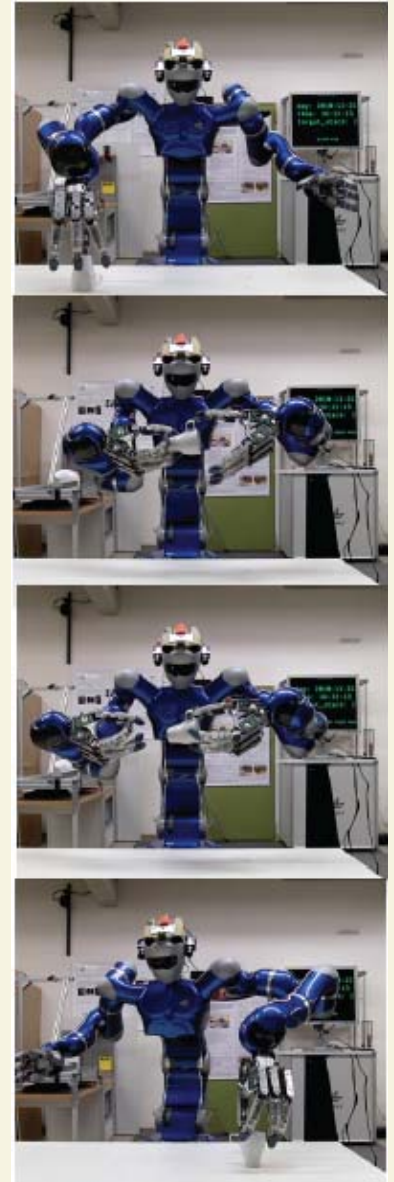
- Determine Similarities between objects

Learning

- Adapt manipulation to new objects

Integration

- Combine all and run it on a real robot

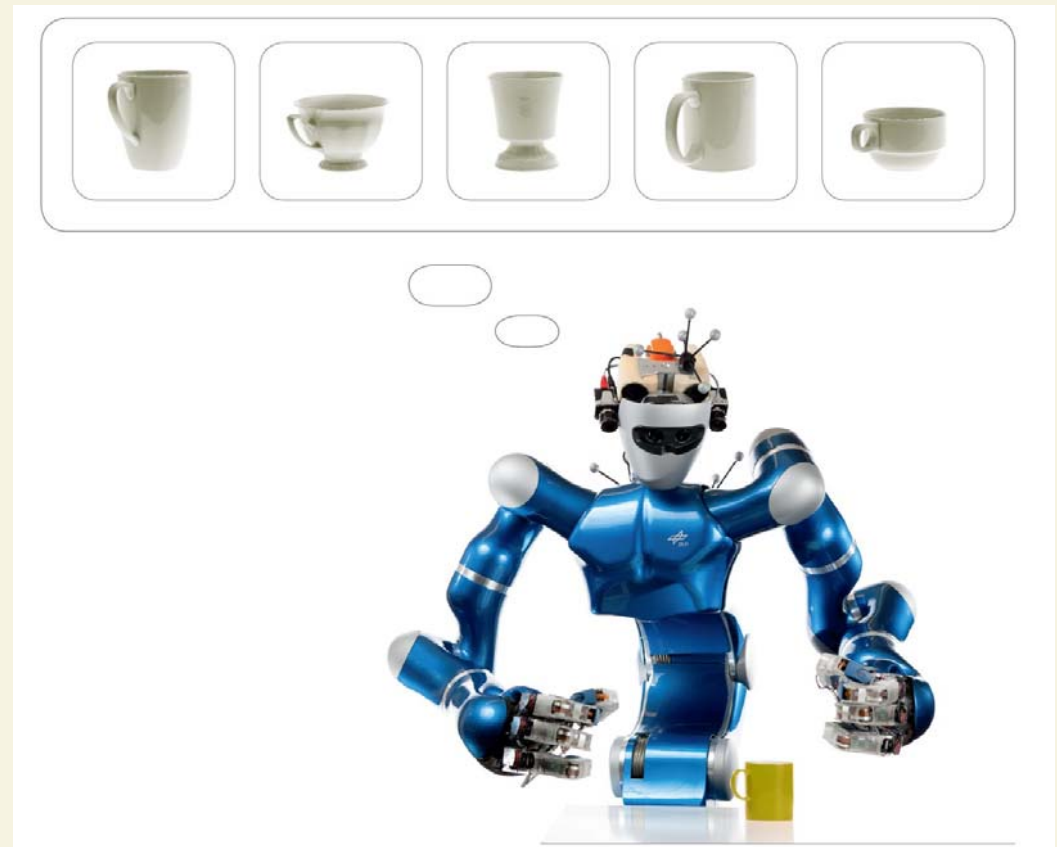


Research Areas

- **Hybrid planning of robotic tasks**
 - Logical planning is required to generate plans from abstract tasks
 - Planning operators are to be learned from example programs
 - Geometric states have to be accounted to generate executable plans
- **Visual perception: recognising objects, parts, affordances**
 - Functional object classes have to be recognized and labeled
 - Correspondences between functional parts of known objects and new objects have to be found.
- **Learning and Optimizing Grasp Strategies**
 - Grasps from the example tasks have to be mapped and adapted to the changed geometry of new object (based on the perceptual information)
 - Reach to grasp strategies (control schemes) have to be developed that allow adaptation and preserve robustness

Integration / Example Programs

- **What are good example programs?**
- **How is the structure of example programs?**
- **How can we support parallel development/integration?**



What are good example programs?

- The GeRT inspiring “Instant tea preparation”
- control program is a sequence of actions given in an XML file
- end of interpolation gives stop condition
- through moderate stiffness trajectories are **implicitly** adapted.



The tea preparation




Good example programs?

Control programs need:

- stop conditions based on sensor input
- control structures like if-clauses, loops, switch-statements
- state-machines to control movements
- to be online interpreted to be fed by hybrid planners

The structure of example programs

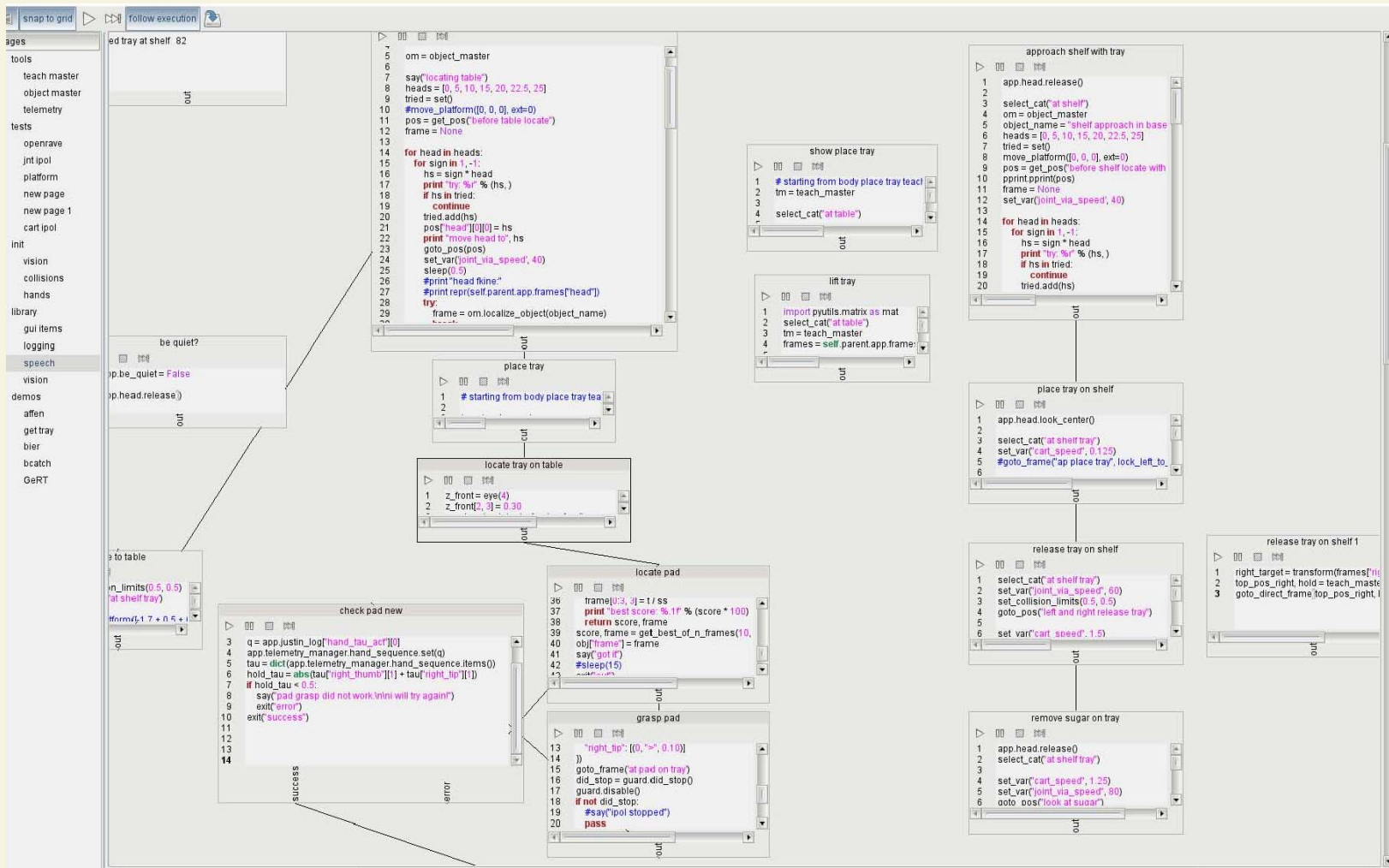
- GUI code displays python source (pyrs-source)
- Script Server code provides a general set of robot commands (macros)
- pyrs-code runs in GUI process, can be single stepped



```

lift object
36 goto_name("at sugar" + pos_name)
37 enable_log(25000)
38 reset_log()
39 # grasp
40 annotate_log("grasp")
41 goto_pos("grasp sugar" + pos_name, vias=["pre grasp sugar" + pos_name])
42
43 # lift
44 annotate_log("lift")
45 goto_frame("approach sugar" + pos_name)
46
47 # wait
48 annotate_log("wait")
49 sleep(0.5)
50 torques = app.justin_log["hand_tau_act"][0]
51 right_hand = norm(torques[:12])
52 print "right hand torque norm: %.2fNm" % right_hand
53 if right_hand < 0.20:
54     log_base += "_failed"
55
56 # place
57 annotate_log("place")
58 if with_guard:
59     guard.enable({
60         "right_arm": [(0, '<', 0)].
61     })
62 goto_frame("at sugar" + pos_name)
63 if with_guard:
64     guard.disable()
65
66 # leave
67 annotate_log("release")
68 goto_pos("open right hand", vias=["pre grasp sugar" + pos_name])
69
    
```

The structure of example programs



The screenshot displays a software development environment with a main code editor and several sub-program windows. The main editor shows a Python script with the following structure:

```

4
5 om = object_master
6
7 say("locating table")
8 heads = [0, 5, 10, 15, 20, 22.5, 25]
9 tried = set()
10 #move_platform([0, 0, 0], ext=0)
11 pos = get_pos("before table locate")
12 frame = None
13
14 for head in heads:
15     for sign in 1, -1:
16         hs = sign * head
17         print "by: %r" % (hs,)
18         if hs in tried:
19             continue
20         tried.add(hs)
21         pos["head"][0] = hs
22         print "move head to", hs
23         goto_pos(pos)
24         set_var("joint_via_speed", 40)
25         sleep(0.5)
26         #print "head fline"
27         #print repr(self.parent.app.frames["head"])
28         try:
29             frame = om.localize_object(object_name)
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
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
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
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

The sub-program windows show the following code:

- show place tray:**

```

1 # starting from body place tray tea
2 tm = teach_master
3
4 select_cat("at table")

```
- lift tray:**

```

1 import pyutils.matrix as mat
2 select_cat("at table")
3 tm = teach_master
4 frames = self.parent.app.frame:

```
- approach shelf with tray:**

```

1 app.head.release()
2
3 select_cat("at shelf")
4 om = object_master
5 object_name = "shelf approach in base
6 heads = [0, 5, 10, 15, 20, 22.5, 25]
7 tried = set()
8 mova_platform([0, 0, 0], ext=0)
9 pos = get_pos("before shelf locate with
10 print print(pos)
11 frame = None
12 set_var("joint_via_speed", 40)
13
14 for head in heads:
15     for sign in 1, -1:
16         hs = sign * head
17         print "by: %r" % (hs,)
18         if hs in tried:
19             continue
20         tried.add(hs)

```
- place tray on shelf:**

```

1 app.head.look_center()
2
3 select_cat("at shelf tray")
4 set_var("cart_speed", 0.125)
5 #goto_frame("ap place tray", lock_left_to_
6

```
- release tray on shelf 1:**

```

1 right_target = transform(frames["tr
2 top_pos_right, hold = teach_maste
3 goto_direct_frame_top_pos_right, l

```
- remove sugar on tray:**

```

1 app.head.release()
2 select_cat("at shelf tray")
3
4 set_var("cart_speed", 1.25)
5 set_var("joint_via_speed", 80)
6 goto_pos("look at sugar")

```
- locate pad:**

```

36 frame[i][j], j] = 1 / ss
37 print "best score: %1r" % (score * 100)
38 return score, frame
39 score, frame = get_best_of_n_frames(10,
40 obj["frame"]) = frame
41 say("got it")
42 #sleep(15)
43

```
- grasp pad:**

```

13 "right_tip": [(0, "x", 0.10)
14 ]
15 goto_frame("at pad on tray")
16 did_stop = guard.did_stop()
17 guard.disable()
18 if not did_stop:
19     #say("ipol stopped")
20     pass

```
- check pad new:**

```

3 q = app.justin_log["hand_tau_act"][0]
4 app.telemetry_manager.hand_sequence.set(q)
5 tau = dict(app.telemetry_manager.hand_sequence.items())
6 hold_tau = abs(tau["right_thumb"][1] + tau["right_top"][1])
7
8 if hold_tau < 0.5:
9     say("pad grasp did not work. hini will try again")
10     exit("error")
11
12
13
14

```

The console at the bottom shows the following output:

```

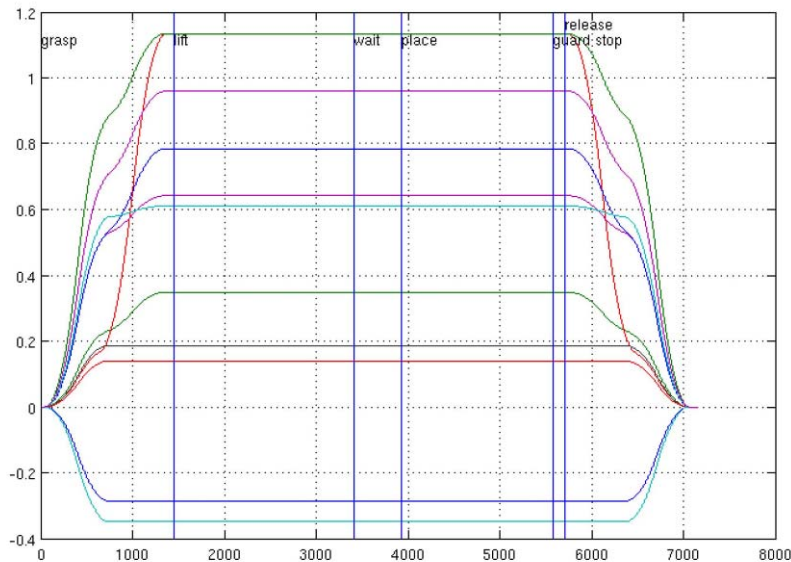
success
error

```

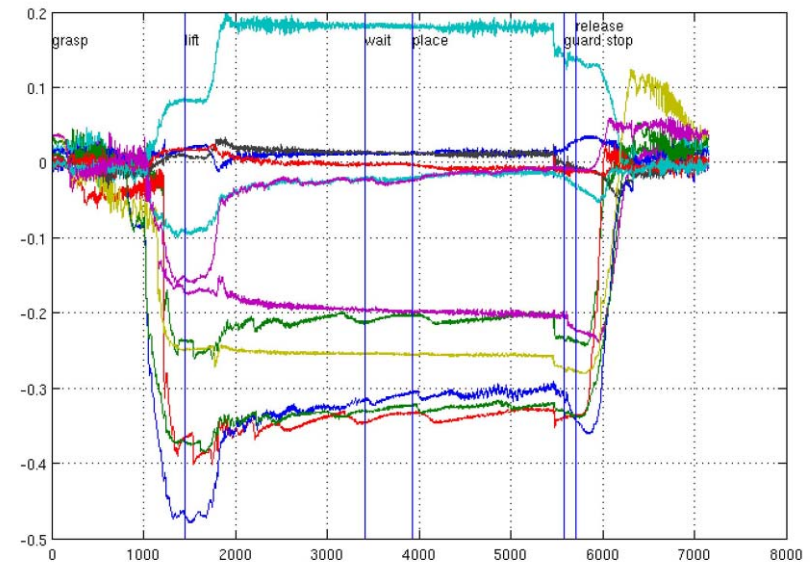

Example: Stacking cups



Example: Stacking cups / sensor log



commanded joint positions

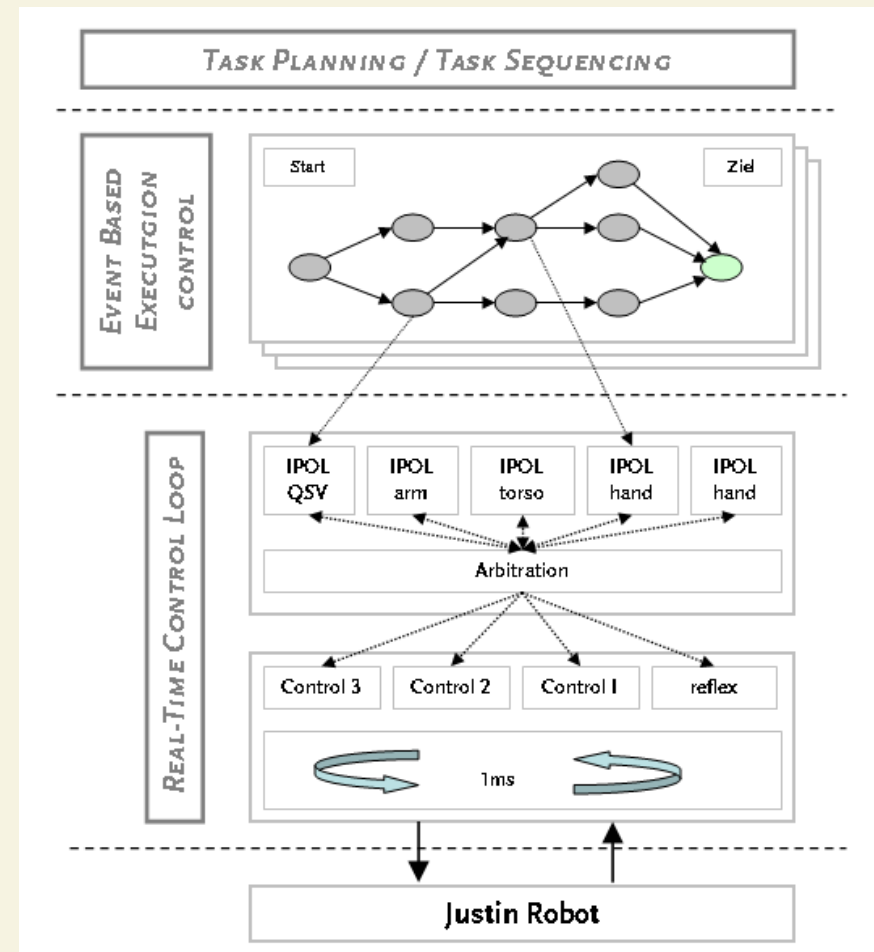


measured joint torques

Sensor logs are annotated in relation to the example programs

Justin's execution environment

- Clear separation of real-time feedback control loop, event based execution control and higher planning levels
- Immediate response on external disturbances through real-time guards.
- Compliant control modes of hands, arms and torso with variable stiffness and damping



HYBRID PLANNER

HTN2 Planner
JShop

PRM Planner
PathPlanning

Justin
Interface (TCP)

TOP Gui

operation editor

task and operation
monitor

telemetry data

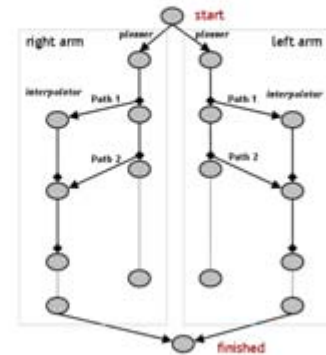
start & stop

VERBOSE (VERBOSE ROBOT SCRIPTING ENVIRONMENT)

pages
(holds items)
Container for
execution
environments

items
gui code (local)
control code
(remote)

pyrs - item
control code
gui code (local)
stubexec code
(remote)



TASK ORIENTED PROGRAMMING (TOP)

XML
Operation
Database

Operation
Sequencing

Control
Execution
Monitoring

TOP SCRIPT SERVER

Start, Stop
Continue
Send XML Script

RT CONTROL SYSTEM

Cart. ipol
left arm

Cart. ipol
right arm

ipol
hand

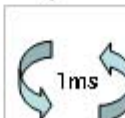
ipol
torso

ipol
all joints

qsv-ipol
all joints

consistent control switching

external
disturbance
observer



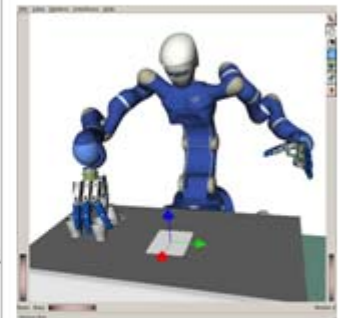
real time
controller

guard
functions

commands

sensor data
(position/torque)

VISUALIZATION OPEN RAVE



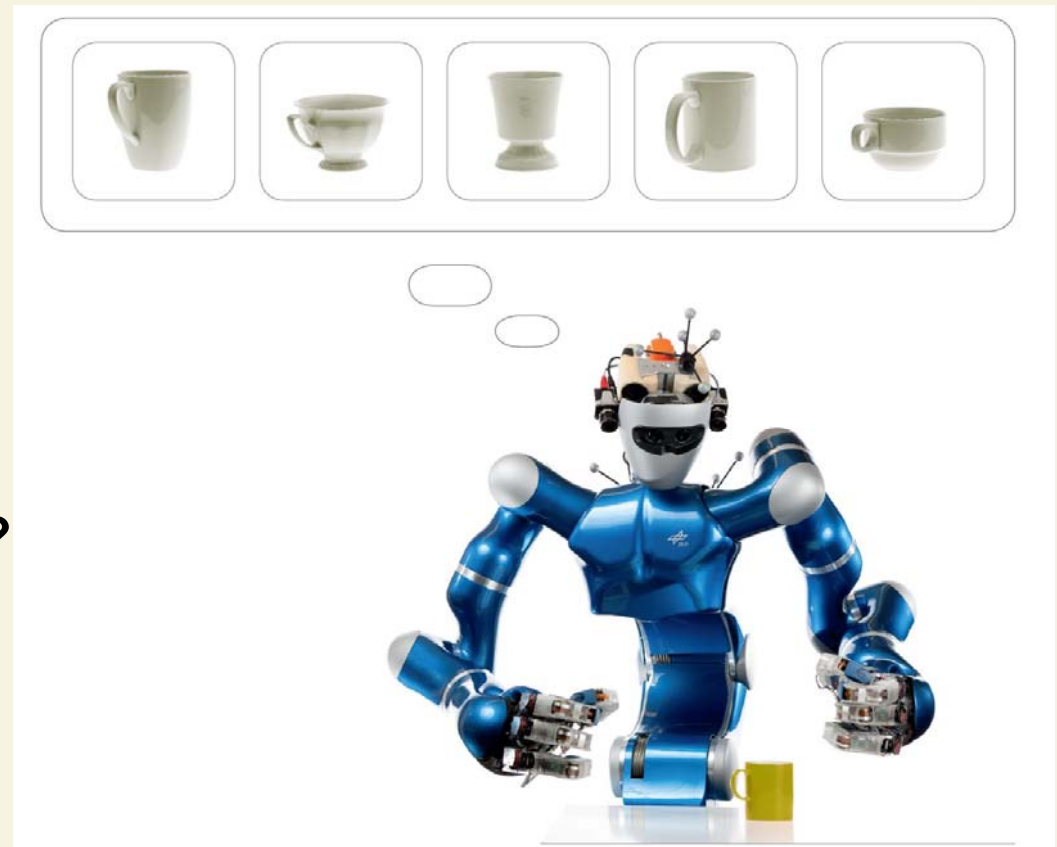
VIRTUAL ENVIRONMENT

collision check

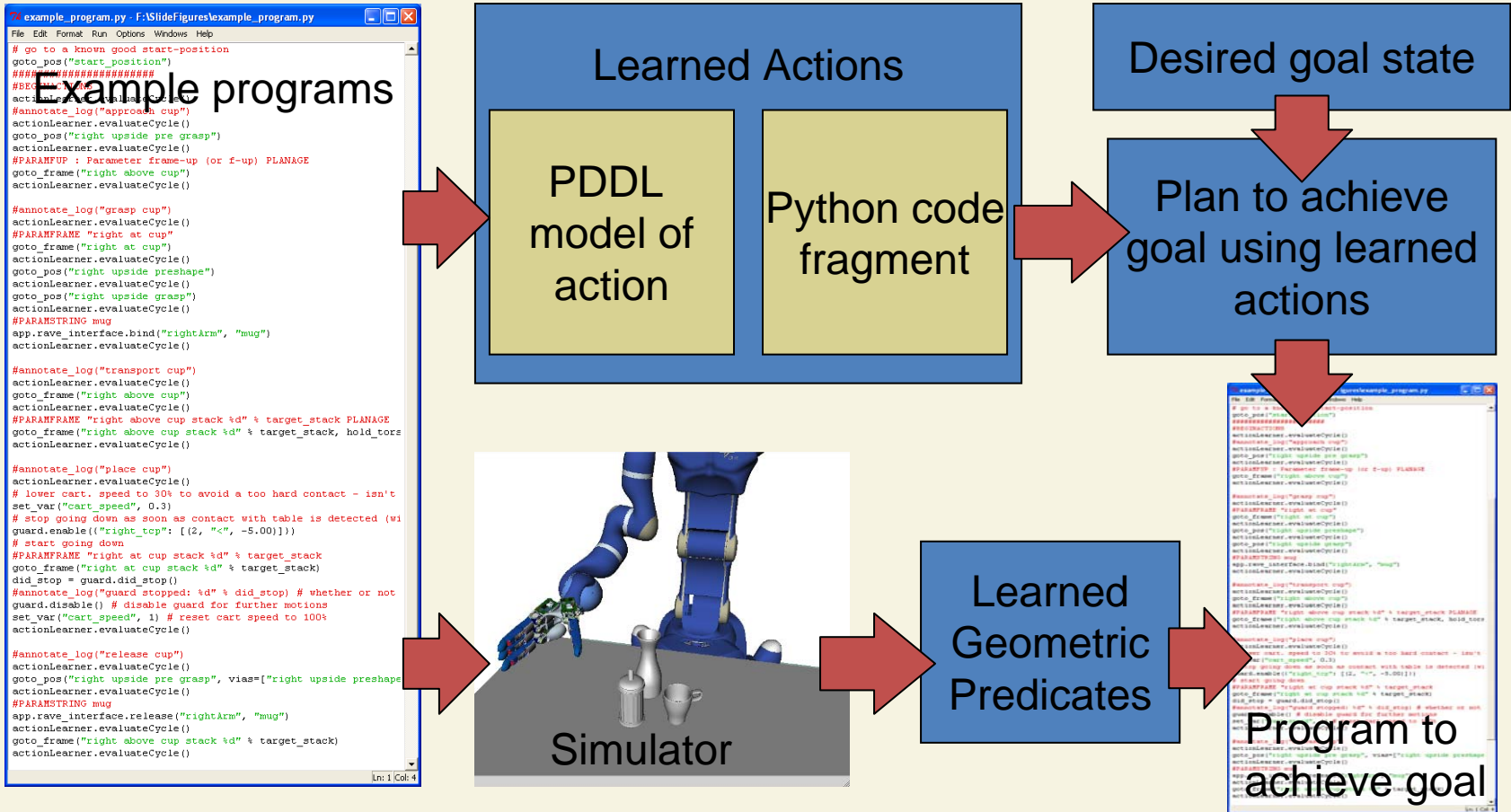
virtual torques

Action generalisation / hybrid planning

- **How can we analyse / generalise the examples?**
- **How are the general actions used to replan the same semantic task?**



Action learning overview



Need to learn action models, but also mapping between geometric predicates (“on-top-of”, “touching”, ...) and geometric states.

Action learning

```

example_program.py - F:\SlideFigures\example_program.py
File Edit Format Run Options Windows Help
# go to a known good start position
goto_pos("start position")
#####
#BEGINACTIONS
actionLearner.evaluateCycle()
#annotate_log("approach c")
actionLearner.evaluateCycle()
goto_pos("right upside pre")
actionLearner.evaluateCycle()
#PARAMFRAME "right above c"
goto_frame("right above c")
actionLearner.evaluateCycle()
#annotate_log("grasp cup")
actionLearner.evaluateCycle()
#PARAMFRAME "right at cup"
goto_frame("right at cup")
actionLearner.evaluateCycle()
goto_pos("right upside preshape")
actionLearner.evaluateCycle()
goto_pos("right upside grasp")
actionLearner.evaluateCycle()
#PARAMSTRING mug
app.rave_interface.bind("rightArm", "mug")
actionLearner.evaluateCycle()
#annotate_log("transport")
actionLearner.evaluateCycle()
goto_frame("right above c")
actionLearner.evaluateCycle()
#PARAMFRAME "right above c"
goto_frame("right above c")
actionLearner.evaluateCycle()
#annotate_log("place cup")
actionLearner.evaluateCycle()
# lower cart speed to 30
set_var("cart_speed", 0.3)
# stop going down as soon
guard.enable(("right_arm"))
# start going down
#PARAMFRAME "right at cup"
goto_frame("right at cup stack %d" % target_stack)
did_stop = guard.did_stop()
#annotate_log("guard stopped: %d" % did_stop) # whether or not
guard.disable() # disable guard for further motions
set_var("cart_speed", 1) # reset cart speed to 100%
actionLearner.evaluateCycle()

#annotate_log("release cup")
actionLearner.evaluateCycle()
goto_pos("right upside pre grasp", vias=["right upside preshape"])
actionLearner.evaluateCycle()
#PARAMSTRING mug
app.rave_interface.release("rightArm", "mug")
actionLearner.evaluateCycle()
goto_frame("right above cup stack %d" % target_stack)
actionLearner.evaluateCycle()
    
```

```

#annotate_log("grasp cup")
actionLearner.evaluateCycle()
#PARAMFRAME "right at cup"
goto_frame("right at cup")
actionLearner.evaluateCycle()
goto_pos("right upside preshape")
actionLearner.evaluateCycle()
goto_pos("right upside grasp")
actionLearner.evaluateCycle()
#PARAMSTRING mug
app.rave_interface.bind("rightArm", "mug")
actionLearner.evaluateCycle()
    
```

PDDL Representation of Action

```

(:action learned_act_1 :parameters ( ?param0 )
 :precondition (and
  ( forall (?ALL)( not (above ?ALL ?param0)))
  (right-hand-empty)
  (left-hand-empty)
  (object ?param0)
 )
 :effect (and
  (right-can-grasp ?param0)
  ( forall (?ALL)( not (right-can-grasp ?ALL)))
 )
    
```

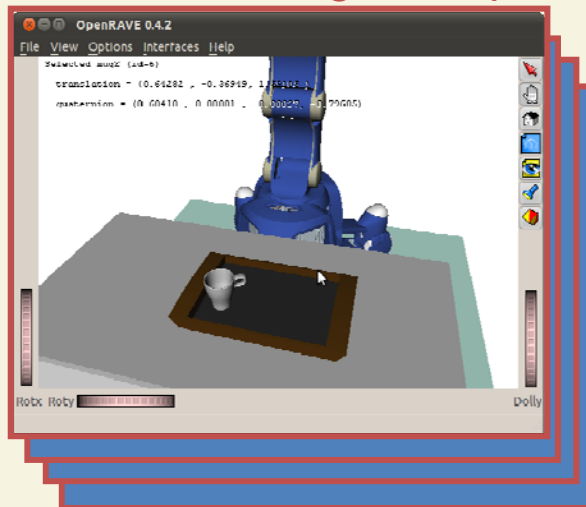
← Unnecessary precondition

- Planning domain definition language action representation learned from annotated code
 - Annotations split up actions, tell us what the parameters of the actions are
 - Preconditions and effects learned by generalising over state before and after code is executed

Geometric Predicate Learning

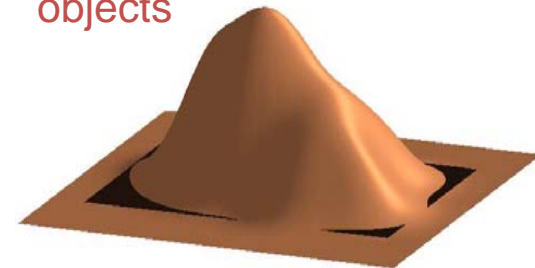
- As well as actions, we need to learn predicates for the planning domain and their corresponding geometric states
 - Do this using training data that could be generated from the example programs
 - At present we augment this with hand-generated data

Labelled Training Examples



Learned geometric relation

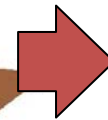
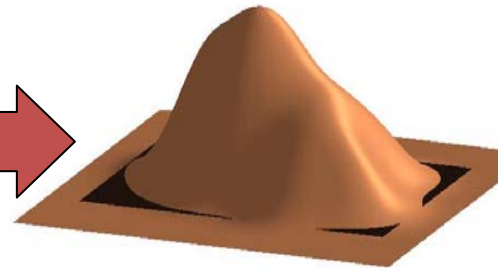
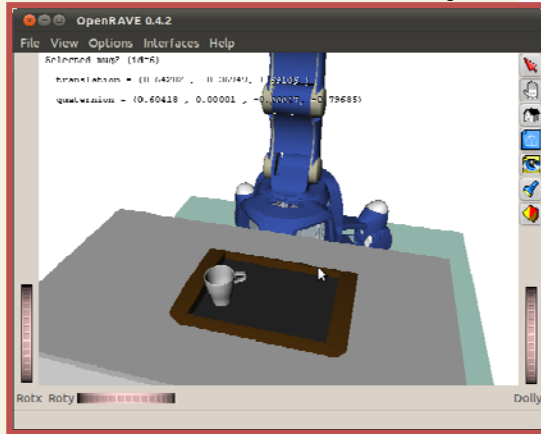
- Mixture of Gaussians model
- 17 dimensional model of relationship between two objects



$P(\text{Above}(\text{cup}, \text{tray}) = \text{true})$. Usable tray shown in black (two dimensions shown).

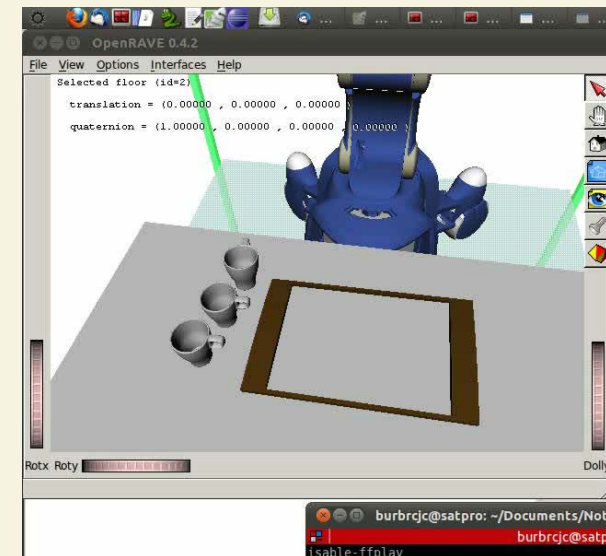
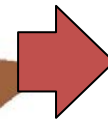
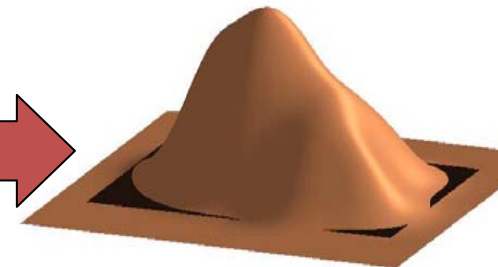
Using the learned model

Forward direction: just query distribution



$$P(\text{on}(\text{cup}, \text{tray}) = \text{true}) = 0.95$$

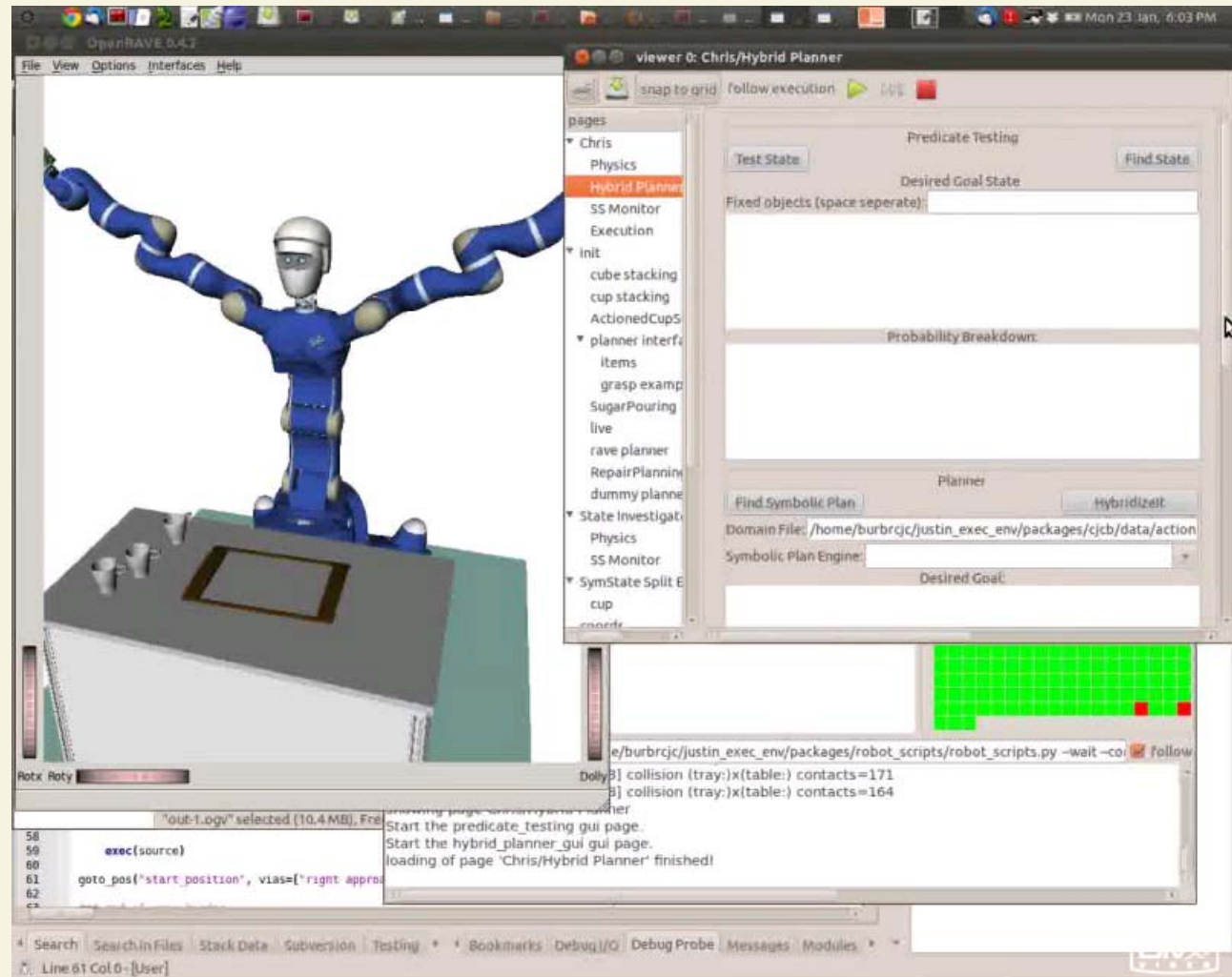
On(cup1, tray),
On(cup2, tray),
On(cup3, tray)



Backward direction: hill-climbing search in probability space

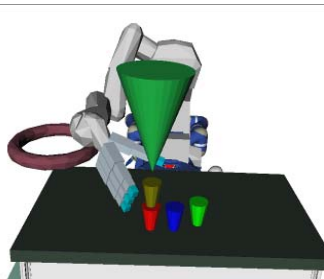
Putting it all together

1. Goal specified to planner
2. Planner generates symbolic plan
3. Learned predicates used to get geometry
4. Code generated and executed

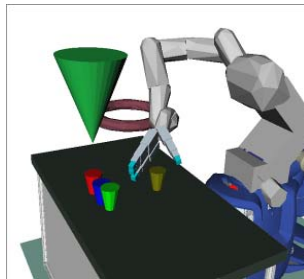


Verfying the plan through hybrid planner

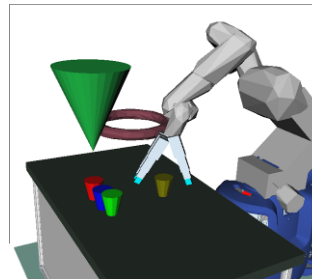
- The symbolic part of the plan can be generated
- Geometric predicates and actions are learned
- Geometric state is evaluated trough RRT planner



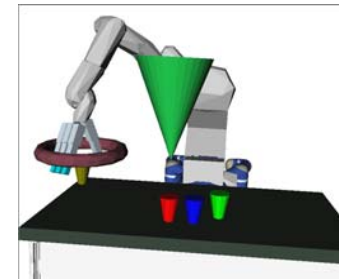
PICK Cup4



**PLACE Cup4
ON Table**



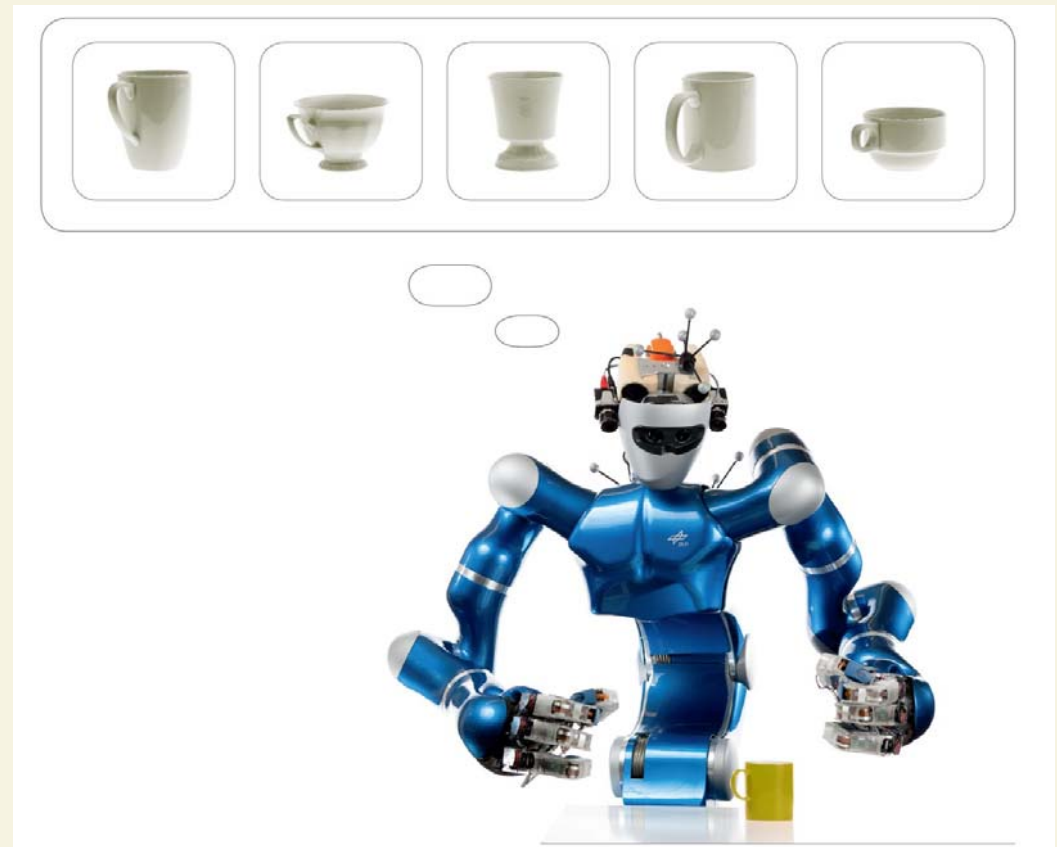
PICK Cup4



**PLACE Cup4
ON Torus**

Perception

- What are the objects of the same functional object class?
(Shape Similarity)
- How correspond functional geometric parts of known and unknown objects?
(Shape Warping)



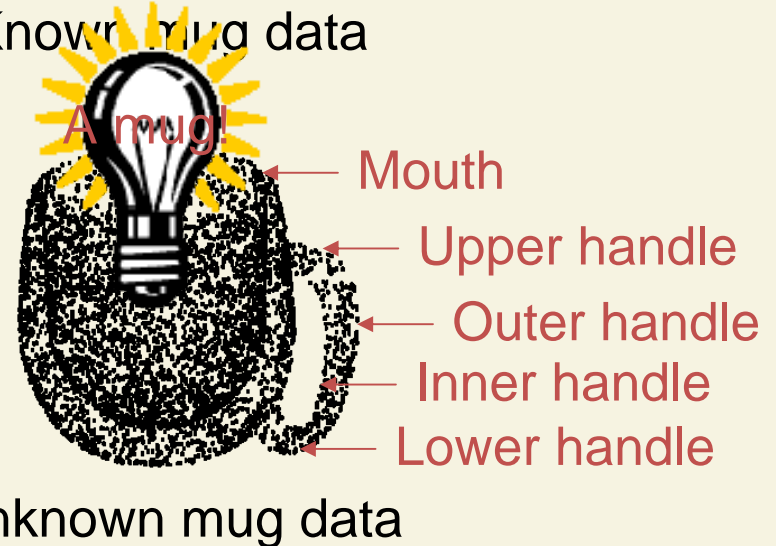
Perception in GeRT:

Understanding Unknown Objects for Manipulation



Rigid alignment

Non-rigid alignment

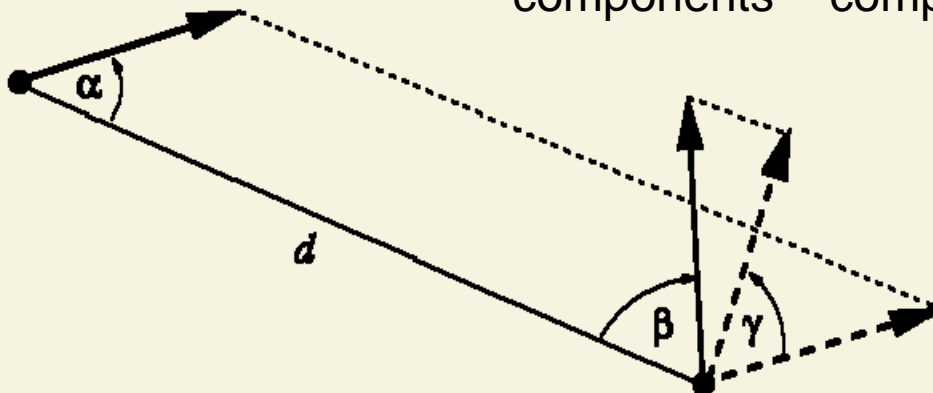


Shape Similarity: Geometric Feature

For query and model shapes independently:

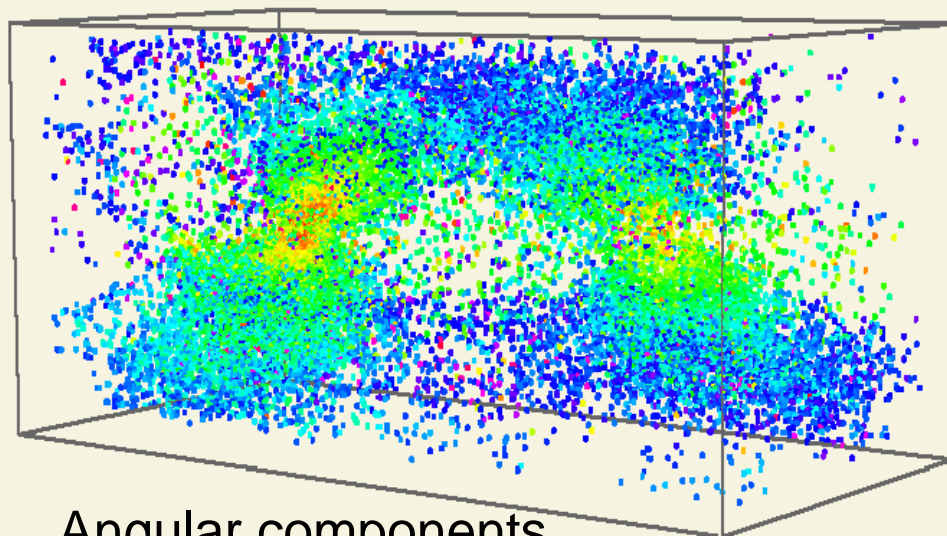
- drawing of random *surflet pairs*
- computation of intrinsic geometric parameters of each surflet pair: $(\alpha, \beta, \gamma, d) \in \mathbf{R}^4$

Angular components Distance component

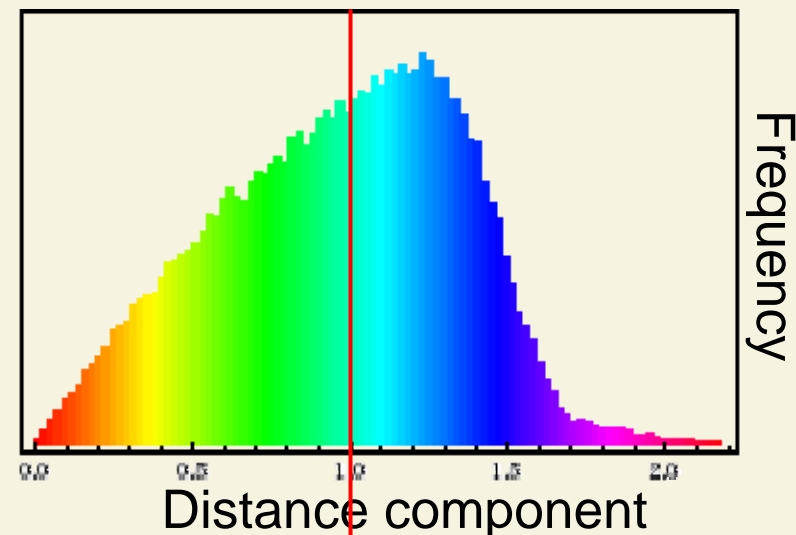


Shape Similarity: 4D Feature Density

For each query/model view: random sampling of 100,000 surflet pair relations



Angular components



Median distance
scaled to unity

Shape Retrieval: Example



Query: Mug

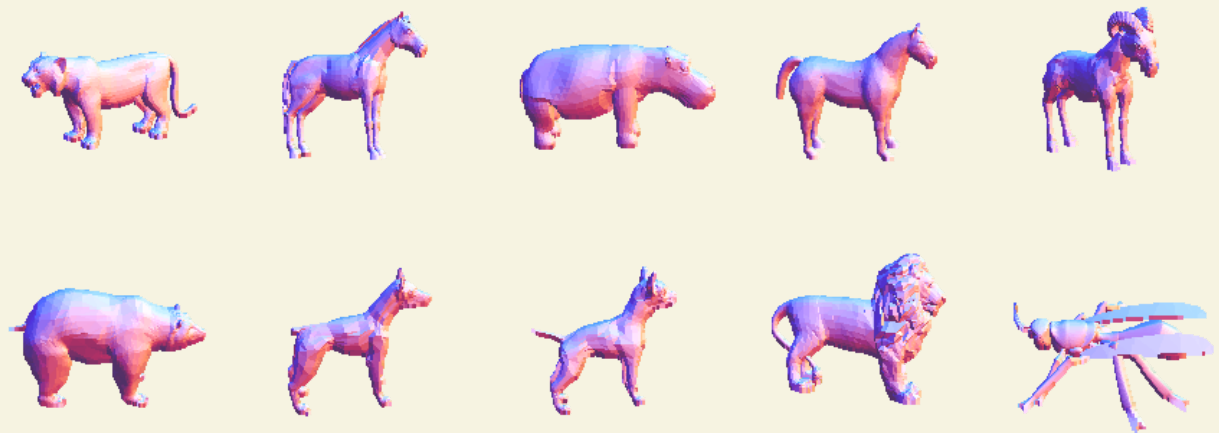


20 most similar models from the database

Shape Retrieval: Example



Query: Quadruped



20 most similar models from the database

Shape Warping

Given a *source shape (model)* and a *target shape (scene)* from the same functional category...



Four stages of warping:

1. Deformation-tolerant alignment
2. Correspondence assignment (forward and backward)
3. Map interpolation (forward and backward)
4. Enforcing forward/backward consistency

Initial alignment brings corresponding parts close to each other – but *without knowing or computing correspondences*

The rest is simply based on proximity of surface points and surface orientation

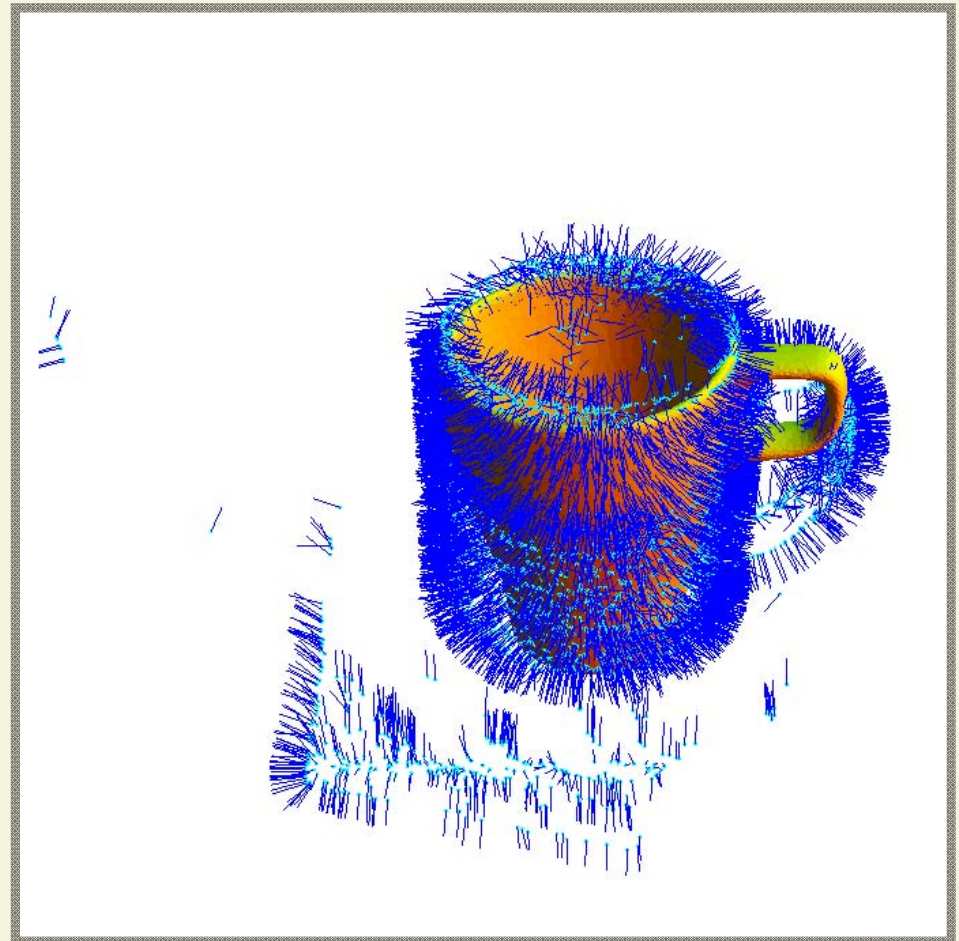
Hillenbrand; ICPR 2010

Shape Warping: Alignment

Source



Target

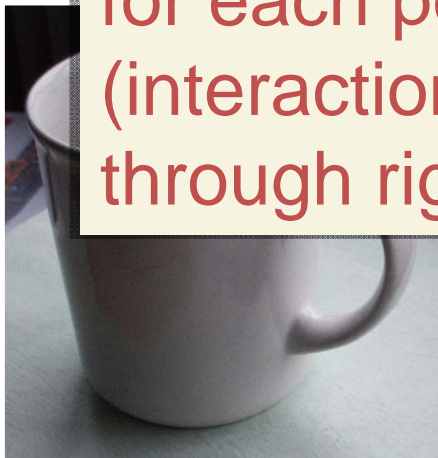


Shape Warping: Correspondence

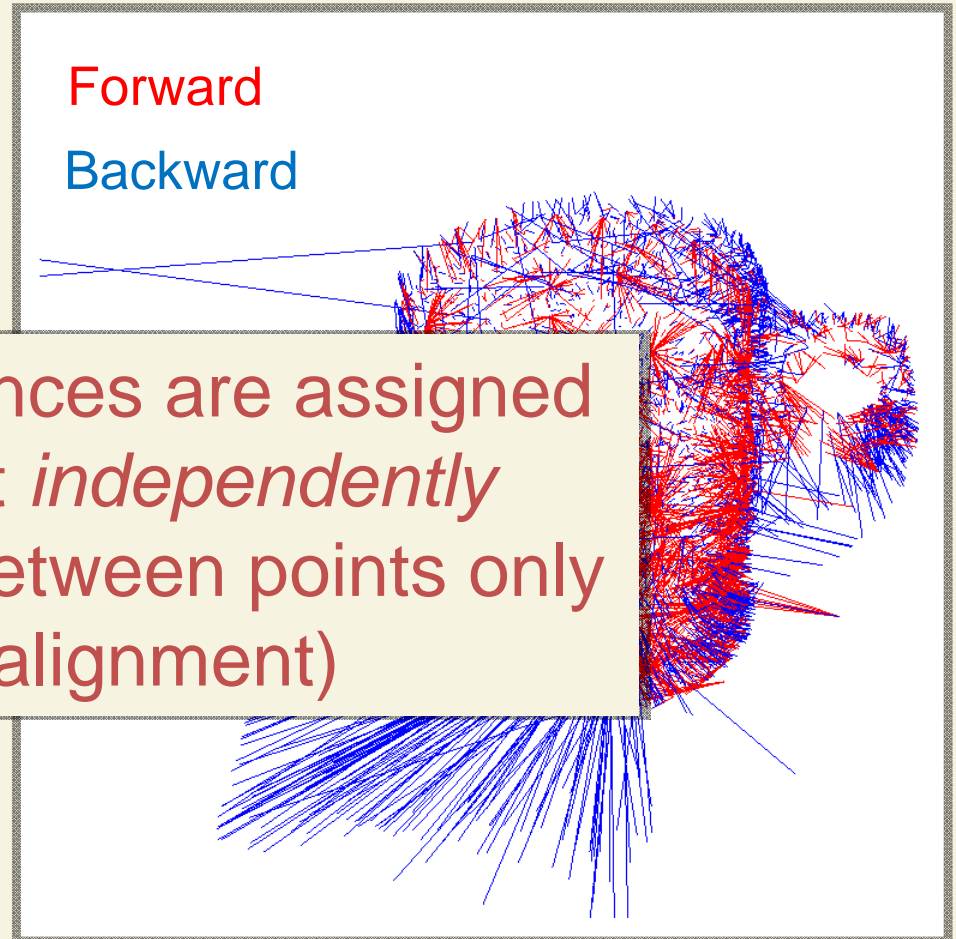
Source



Target

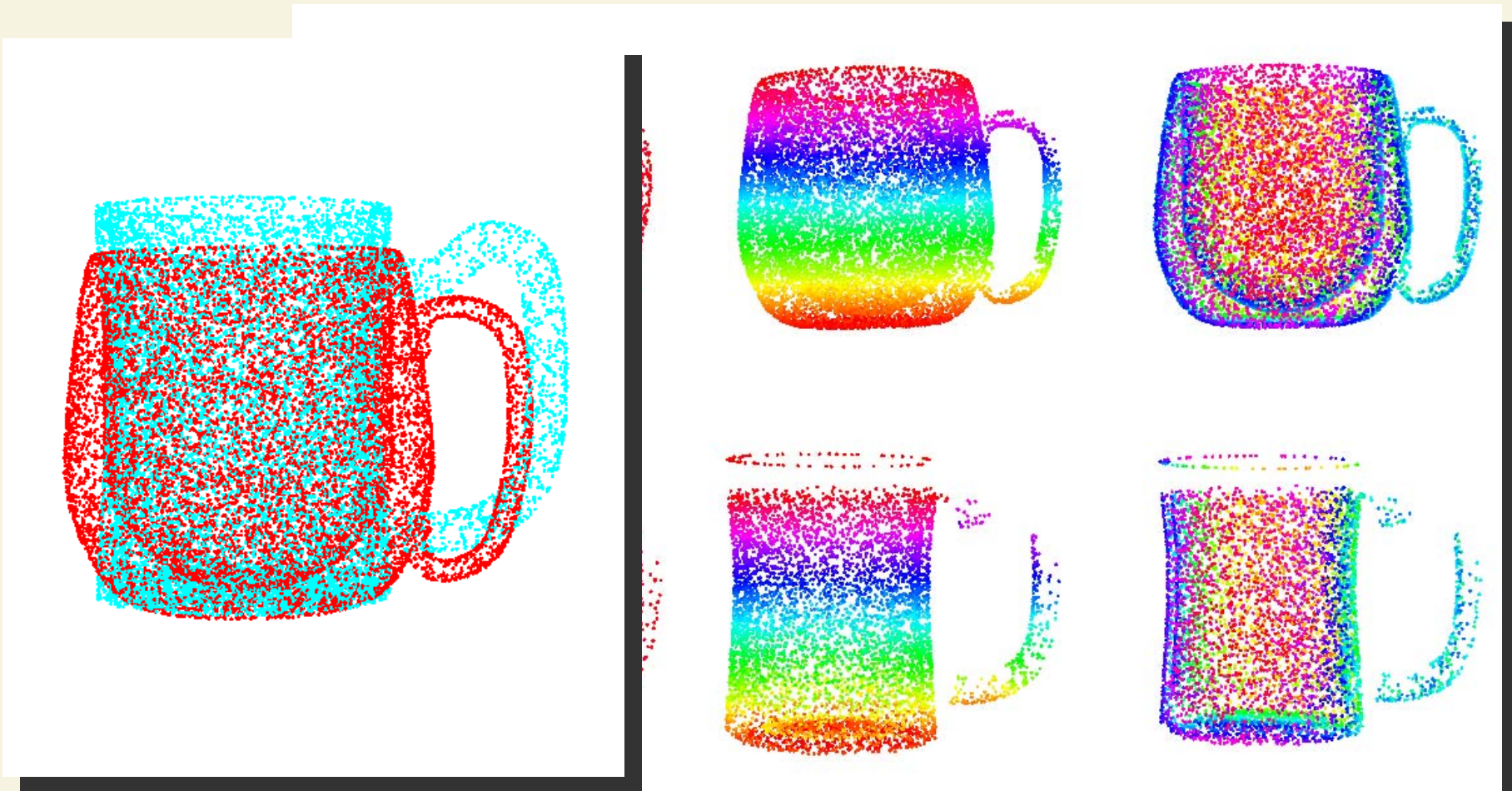


Forward
Backward



Correspondences are assigned for each point *independently* (interaction between points only through rigid alignment)

Shape Warping: Point-Wise

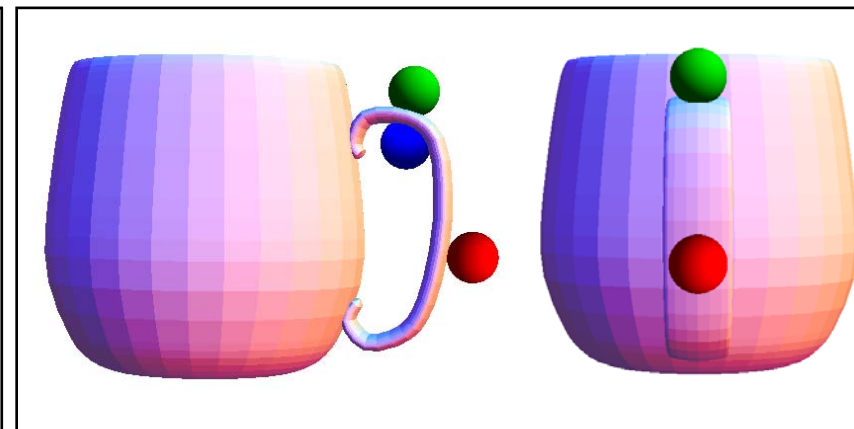
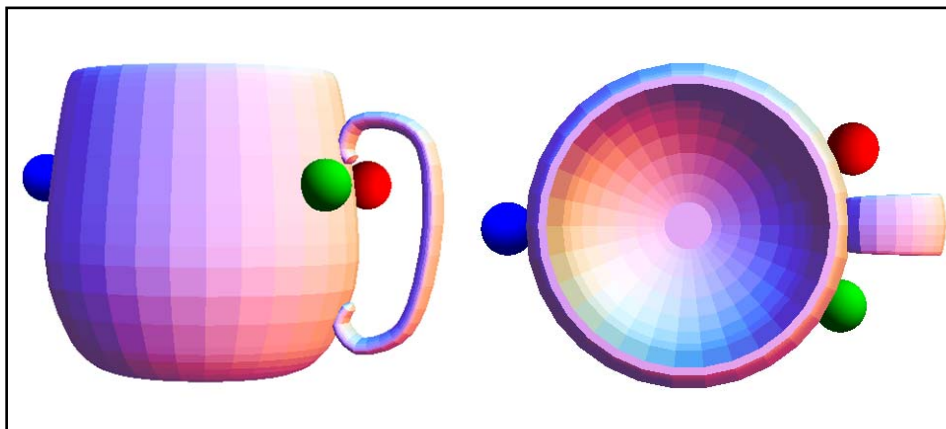


Shape Warping: Grasp Contacts

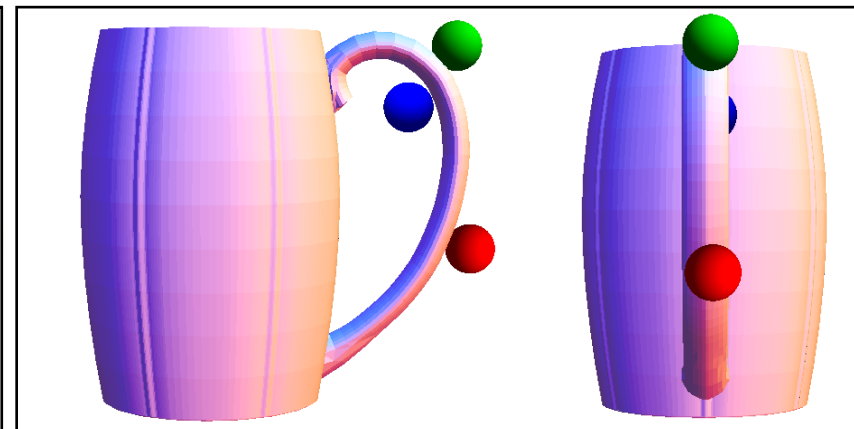
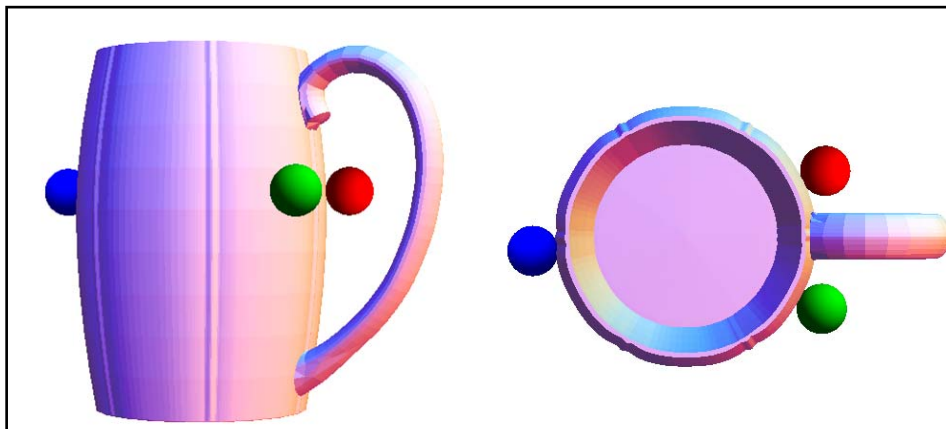
Top Grasp

Handle Grasp

Source

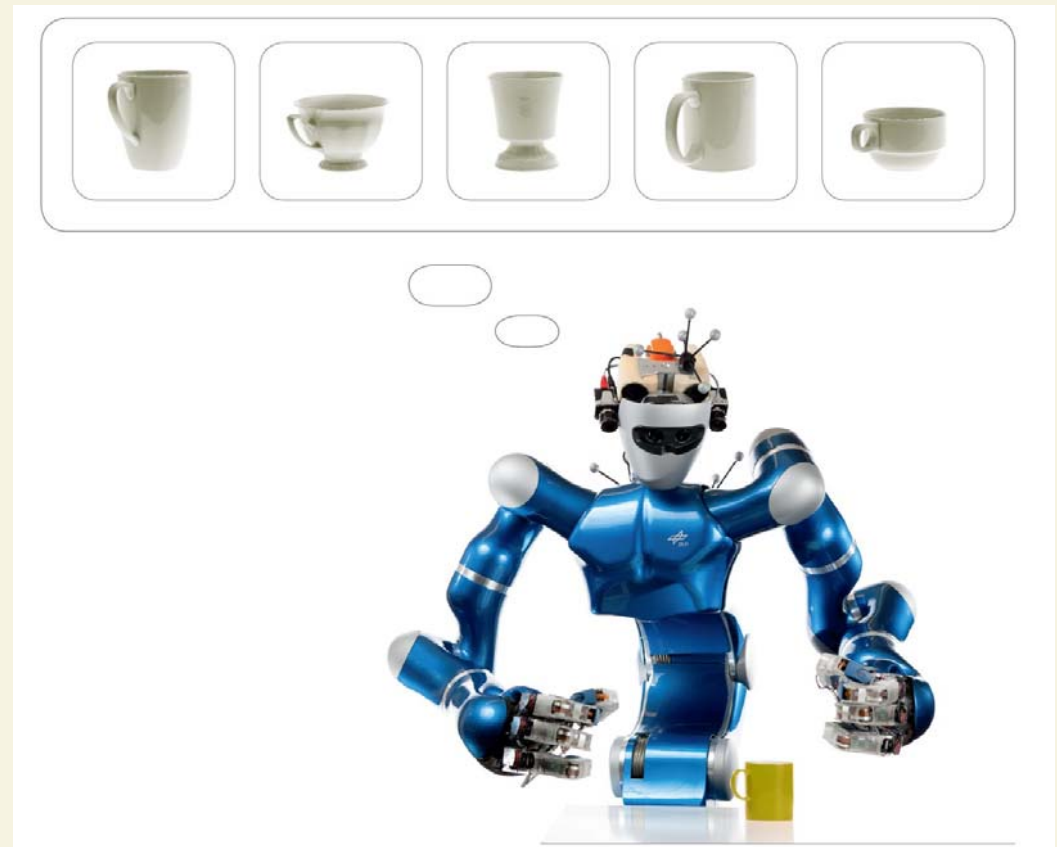


Target



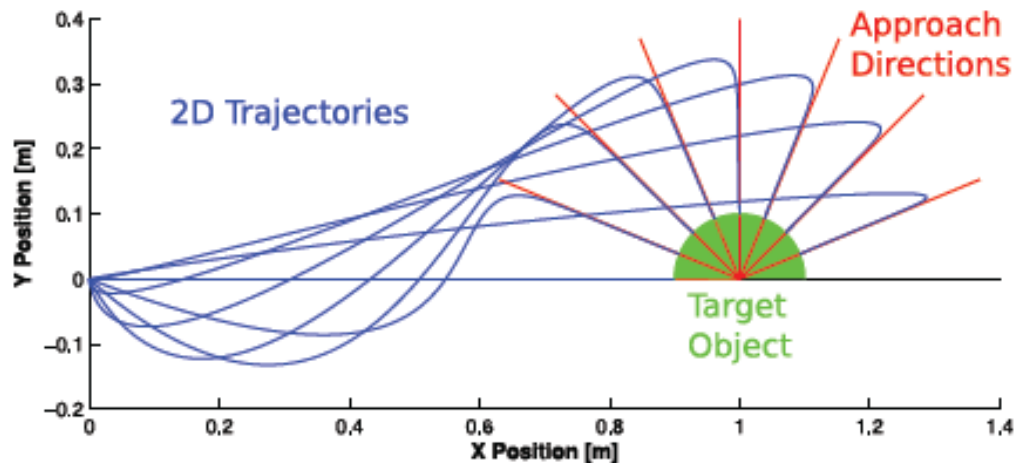
Learning grasping strategies

- Adapting example grasps to new objects
- Make use of human example grasps
- Learn a robust scheme to execute finger strategies



Grasping with Motor Primitives

- Flexible representation for entire grasping action
 - Obtained from human demonstration
 - Automatically adapt to new grasps



Grasp Adaptation

- Generalization of grasps to:
 - New objects
 - New positions and orientations
 - Different grasp types
- 12 Grasp types from our taxonomy
- Grasp types are learned by imitation



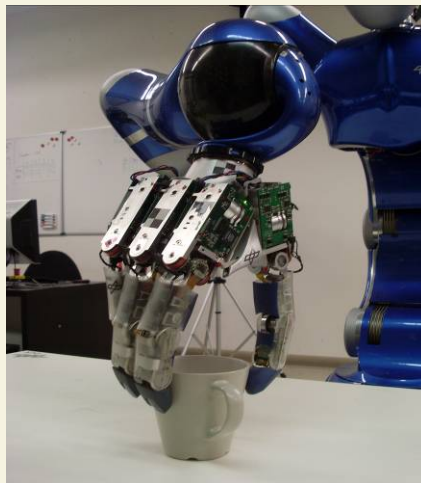
Lateral



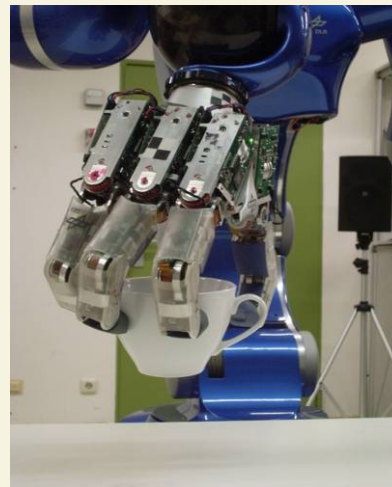
Spherical



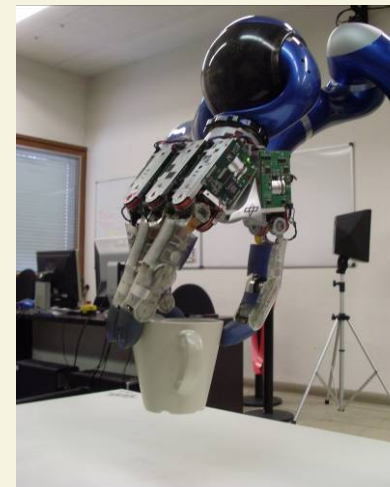
Tripod



Lateral



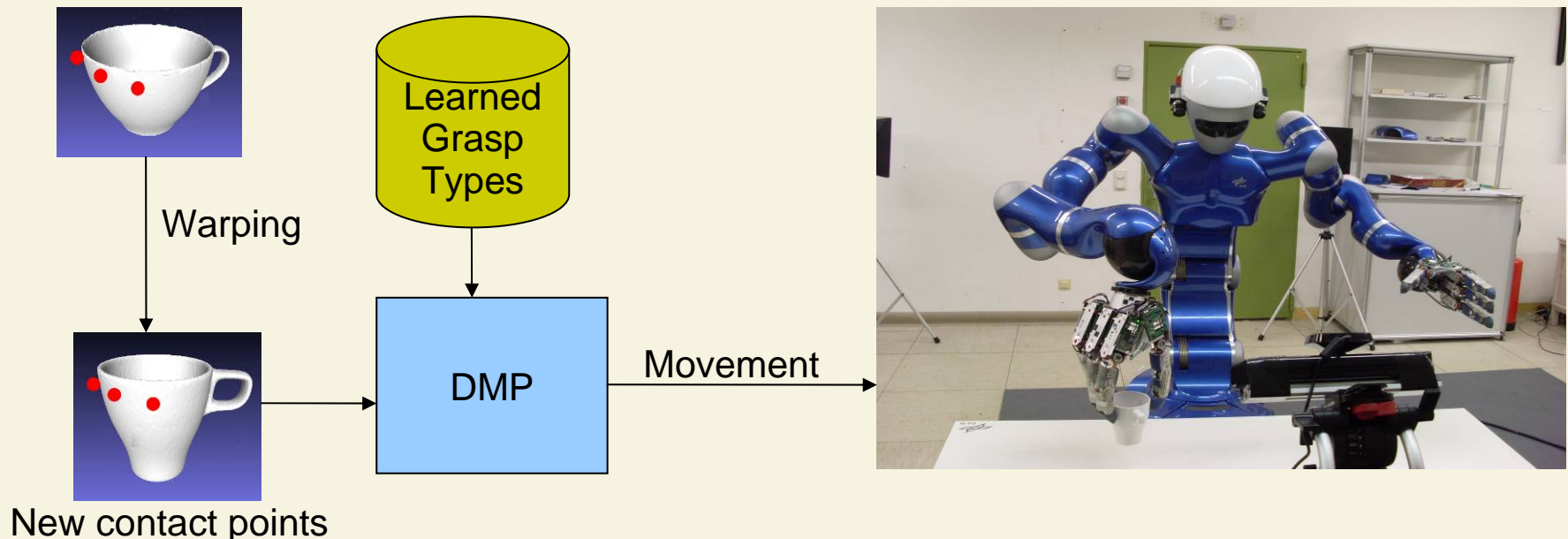
Spherical



Tripod

Imitation by Dynamic Motor Primitives

- Based on Dynamical Systems
- Used to generate Trajectories of arm and hand
- Parametrized by warps of contact points
- Easily generalized to new goal states



Results of last weeks integration week

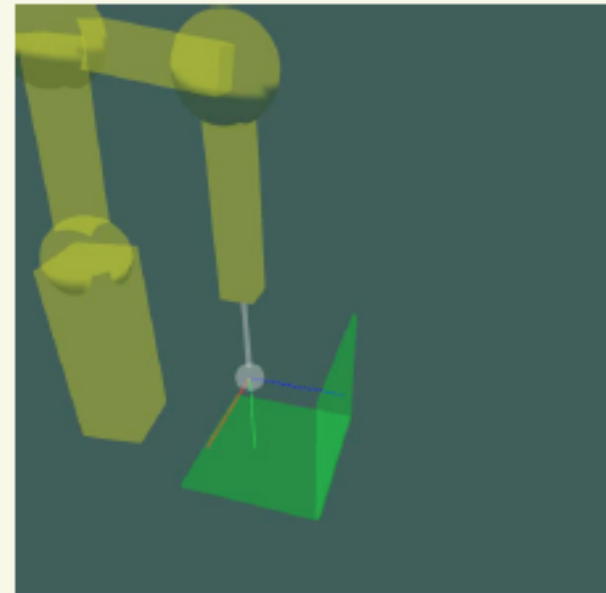
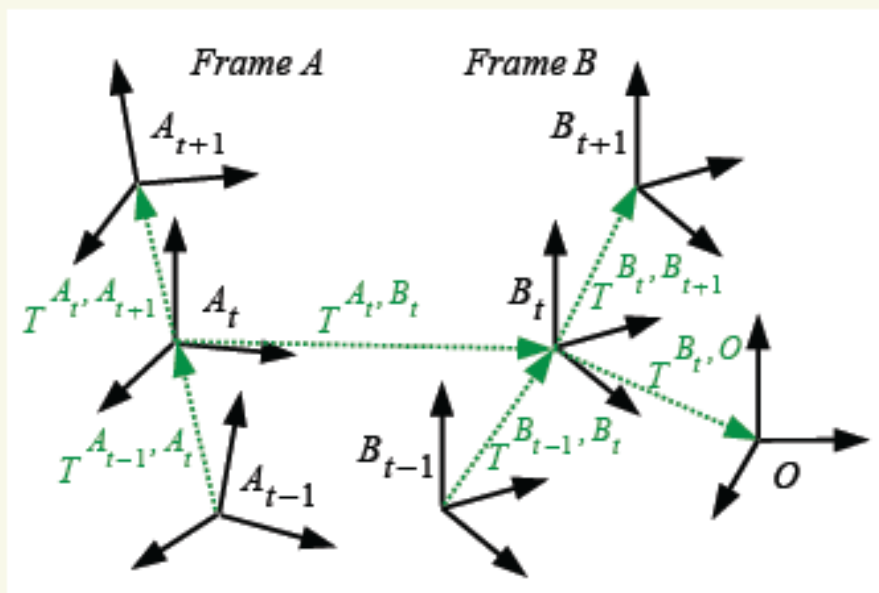


Learning forward models for grasping

- In GeRT we want to generalise from example programs
- Use these to gather data to learn forward models
- Use these to plan actions for novel tasks or objects
- Start with single finger, then multi-finger grasps, in hand manipulation

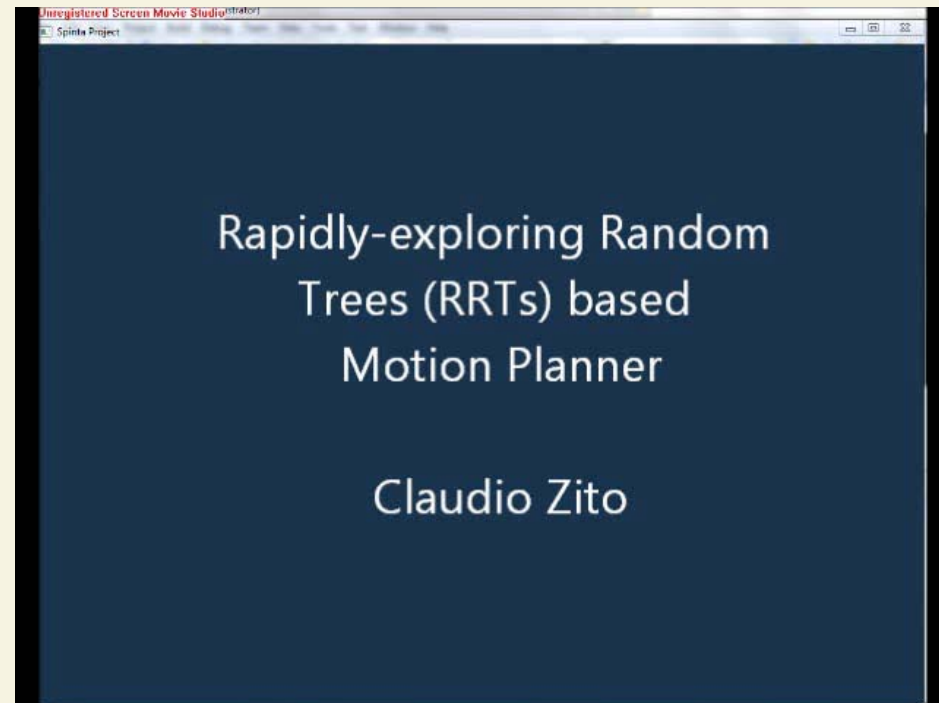


Prediction learning for single contacts



- Robot effector as some frame A
- Object it is pushing as frame B
- Prediction is saying what $B_{t+1:t+N}$ is going to be given $A_{t-1:t+N-1}$, B_t and B_{t-1} (i.e. prediction for N steps into the future)

Prediction learning & planning with prediction models



Next steps:

- Multi-finger manipulation prediction
- Repairing plans that have not been successful
- Adding contact and force information from sensor logs (DMPs)
- Integrating to a working example on Justin

The consortium individuals

