EU FP7 CogX
ICT-215181
May 1 2008 (52months)

DR 9.4:
Proceedings of Spring School 2011

Michael Zillich, Torben Töniges and Marc Hanheide

Vienna University of Technology, University of Birmingham
⟨zillich@acin.tuwien.ac.at⟩

Due date of deliverable:	July 31, 2011
Actual submission date:	July 28, 2011
Lead partner:	TUW
Revision:	final
Dissemination level:	PU

This document describes the CogX Spring School organised at TUW in Vienna, April 28th to May 4th, 2011. This was the final of the three spring schools planned for. The main parts of the school were, invited talks to provide tutorials in areas related to the spring school topic and to the overall CogX theme, technical tutorials covering the use of newly developed system components, and a project to be solved in groups to get hands-on experience and act as a team building activity.

Executive Summary

The third and last CogX Spring School was organised by TUW from April 28th to May 4th, 2011. As with previous CogX Spring Schools the emphasis was on hands-on project work rather than invited talks. In the past this concept has proved to be very successful in driving integration of hard- and software and providing all CogX partners with a focused learning experience in terms of newly developed system components. That was also true this year, where the focus was on mobile manipulation. Spring school participants were organised into four teams to solve the task of retrieving various pre-trained objects and delivering them to a drop-off zone.

The school had four invited speakers giving three lectures. Justus Piater from the Intelligent and Interactive Systems Group at the Institute of Computer Science, University of Innsbruck, Austria and Renaud Detry, from the Computer Vision and Active Perception Lab (CVAP) at the Kungliga Tekniska Högskolan (KTH), Stockholm, Sweden (a CogX team member) presented a tutorial about "Learning to Grasp with Grasp Densities". Horst Bischof from the Institute for Computer Graphics and Vision at TU Graz, Austria presented a tutorial on "On-line learning for computer vision". Finally Fiona McNeill from the DReaM group at the Centre for Intelligent Systems and their Applications, School of Informatics, University of Edinburgh, UK gave a tutorial about "Managing Ontologies: Matching and Repair". While two of the topics were directly related to the tasks of the spring school, the third was of general interest for the approaches taken in CogX.

As in previous schools the members of the four competing teams worked hard, partly late into the night, and at the end of the week had developed impressive systems, with the winning team locating all 3 given objects and successfully grasping and returning two of them to the drop-off zone in the final competition. Results clearly showed the advantages of those teams that put a strong emphasis on extensive integrated testing and intelligent search strategies. This once again showed the importance of well coordinated integration, on software but also on a personal level. To this end the dinners and social event provided opportunities to interact and meet new team members. To conclude, the spring school was very successful.

Role of the Spring Schools in CogX

The CogX project aims not only to contribute new theories but also to implement and create instantiations in robots to test these theories. In CogX the spring schools provide an important vehicle towards this.

The objectives of the CogX Spring Schools include:

- train the researchers in the techniques and tools to be used in the project, and in the methods employed in the state of the art in the

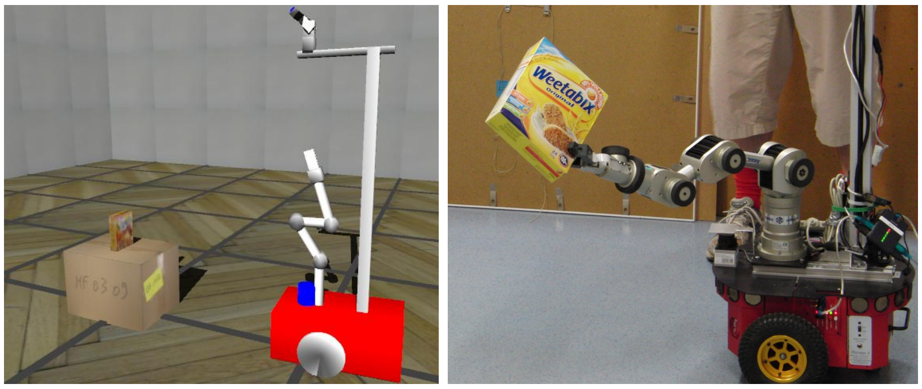


Abbildung 1: Simulated test environment and real robot carrying a cereal box during the final competition.

community

- establish a common ground of theoretical knowledge
- efficiently communicate knowledge to the researchers, both from external parties in the form of invited speakers and from researchers within the consortium
- increase impact of the dissemination by including external parties (invited speakers) in the spring school who get a close look at the project
- build strong connections between the researchers within the consortium by getting together for an extended time, interacting in working and as well as social contexts

Contribution to the CogX scenarios and prototypes

The first spring school introduced the common hardware platform (a Pioneer 3-DX base) and the CAST software framework using a simple search task, thus laying the foundations for the CogX scenarios. The second spring school introduced planning and binding, which are integral parts of the Dora and George scenarios. This year's spring school introduced manipulation, both in terms of new hardware (a Neuronics Katana manipulator) to be mounted on the platforms and in terms of new system functionality. The respective functionality has been used previously within the the Dexter scenario, albeit not integrated into CAST. Now fully integrated, mobile manipulation extends Dora's and George's capabilities, allowing fetching of items or interaction with objects in active learning scenarios. Moreover, a physics based 3D full system simulation was developed as part of the spring school to support extensive testing without needing access to the hardware. This simulation provides a valuable testbed for the Dora and George scenarios, where it is used extensively. It allows testing on a system level on precisely identical systems across the consortium, which greatly simplifies debugging.

1 Tasks, objectives, results

One important ingredient of the well-known "Bring me the cereals" scenario was still missing in the CogX integrated system: the capability to actually grasp objects. This not only entails manipulator path planning as already performed within the Dexter scenario, but also integration in terms of system calibration, consistent coordinate frame representations and high level access to grasping primitives. The main goal of the spring school was to add mobile manipulation capability to the CogX system and familiarise participants with the hardware setup, full system calibration and newly developed software components.

With ever more complex systems testing becomes crucial but also time consuming. Moreover a mobile manipulator requires significantly more supervision, to ensure the arm is not damaged. To this end we also provided a full 3D simulation of the whole system, including stereo vision, laser based navigation and manipulation. The simulation allows frequent testing with no danger of damaging hardware, and proved to be a valuable tool for development.

Two of the invited lectures were chosen to provide theoretical background related to the spring school task: grasp planning and on-line learning in computer vision. The third lecture about ontology matching and repair was chosen for its general relevance to the CogX project.

1.1 Preparations

Hardware: Prior to the spring school all 6 CogX platforms were equipped with identical (subject to minor version differences) Neuronics Katana manipulators as well as Microsoft Kinect RGB-D sensors. Detailed mechanical and electrical mounting instructions were provided and the robots, disassembled for shipping to Vienna, were quickly reassembled.
Software: New low level drivers and high level system components related to manipulation were integrated into the CogX architecture prior to the spring school. Moreover tools and procedures for full system calibration (robot base with laser ranger, manipulator, pan-tilt head, stereo camera system, Kinect) as well as matching calibration patterns were also provided and explained in technical tutorials.

Following the software release schema in CogX the spring school is linked to one of two major annual milestones per year. Consequently, integration efforts peak the weeks before the school and lead to a stable collection of software that is then used by all the participants. The software employed at the CSS 2011 was based on the integration system Dora and George. In addition to the abilities documented in DR7.2 some components have been added:

- an arm controller server, providing low level access to arm functionality

- a manipulation server, providing high level access to collision free manipulator path planning in terms of TCP goal poses, including an easy to use test GUI
- a Player/Gazebo simulation environment including the CogX platform modelled with all sensors and manipulator
- a bridge between manipulator path planning and player to ensure the simulated arm precisely mirrors the movements of the real arm
- software tools for system calibration

To ease implementation and maximise the learning opportunity, a GAR installer repository was set up prior to the school including all required software and the simulation environment. Also prior to the start of the school, participants were asked to install the system and have a few test runs to get used to the general procedure.

1.2 Project Work

Participants, including PIs, were divided into four teams. This meant that two of the 6 available robots were available as spare platforms, should one of team robots break down. The two main factors when forming the groups were i) diversity with regard to the institutions people work for and ii) to distribute the knowledge and skills as evenly as possible among the groups.

The overall task for this spring school was to locate known objects (cereal boxes) in a room, pick them up and deliver them to a drop-off zone. Objects were placed on tables, which were low enough to ensure reachability by the arm. This task was divided into three competitions:

1. "Sweet Stacks" The first competition was intended as an easy and fun introduction to the simulation environment. Participants were asked to take the role of the simulated robot, controlling base position and individual arm joints, and to create stacks of box shaped "Manner" wafers, where higher stacks would score more points. The score in the first competition determined the starting order for the second competition. Besides familiarisation with the new software, this seemingly simple task should make participants aware of the limitations faced by the real robot in the actual task later on, such as a non-holonomic drive system and an arm with only 5 degrees of freedom. These have to be taken into account when planning strategies for search or pickup.
2. "Hand me the Cereals" The second task was to simply pick up a cereal box placed in front of the robot and to hand it to the user. This task was intended as a sort of milestone, to make sure participants could i) get the arm to move to a given position ii) train and recognise an

object iii) calibrate the system to ensure all the various poses match up. Scores were given for detecting, touching, lifting and finally handing over of the object. Again, scoring in this competition determined the starting order in the next competition.

3. “Cleaning the Kitchen”: The final task was to detect 3 of 5 pre-trained cereal boxes in the test arena and deliver them to a drop-off zone. The objects were placed in a graspable position (upright or lying on the side) on 3 low tables, which had unknown positions. The teams had to devise strategies for search, for positioning the robot, for defining good grasp positions and of course for recovering from failures. Each team had a time slot of 30 minutes, including setup time and time for possible restarts. Scores were again given for detecting, touching, lifting and finally returning an object.

Three of the four teams participated in the final task, and two were finally able to return objects. It was interesting to see the strategies employed by the two best teams. One team placed a strong emphasis on relentlessly testing the robot in the arena. again and again. This led to a remarkably robust system, running for hours. The winning teams decisive advantage was their use of the pan-tilt head, which drastically reduced search times. This allowed the winning team to locate all 3 objects and pick up and return 2 of them to the drop-off zone. The team not participating in the final competition was plagued by a particularly difficult to diagnose hardware error combined with an unforeseen shortage of team members. The team participating, but failing to retrieve an object only integrated their whole system very late in the development.

1.3 Lessons Learned

Starting the spring school with a few interesting lectures followed by 5 days of intense team work in small, heterogeneous teams proved to work out well, as it did in previous schools. Starting with the mature software base developed in the previous years ensured that participants could concentrate on the important issues for their task. The newly developed system components were mercilessly stress-tested, and a few remaining bugs were identified and quickly resolved. Following lessons from the previous spring school the given sub-tasks were designed to incrementally lead development to the final task, without too much sidetracking. Despite one team not taking part in the final competition, it was promising to see how well the considerable complexity of the complete system was handled by the participants. The considerable effort put into the system prior to the spring school and the intense learning experience for the participants were a major boost for system integration.

1.4 Relation to the state-of-the-art

Grasping of objects in real world environments, despite being pursued for some time now, e.g. as part of the RoboCup@Home competitions, remains a challenging topic. This includes practical hardware issues (power consumption, weight and reach of the arm) as well as theoretical issues (path and grasp planning, object recognition). The design of the CogX platform, with its arm mounting and superstructure, turned out to be well suited to handle a limited set of such tasks (namely limited by the reach of the arm, which is mounted low). Also the path planning and object detection and tracking methods developed within the CogX project provide a stable basis for future research in that direction.

2 Proceedings

The proceedings are a modified version of the proceedings handed out to the participants of the CogX Spring School. Most of the local information has been removed and some of the information that was only provided online on the CogX intranet has been included.

First, some general information about the spring school is given. Then the competition tasks are presented. The course material provided for the participants on the CogX Wiki, which served as main instructions for work, is included, subject to the constraints in trying to put the interconnected Wiki pages into a sequential order. The final part of the proceedings is composed of three tutorials presented by the invited speakers.

CogX Spring School 2011
Vienna, 28. April – 4. May

COGX

SPRING SCHOOL

28. April - 4. May 2011 - Vienna - Austria

http://cogx.eu

2.1 Overview

Welcome to the 2011 CogX Spring School in Vienna! The following pages will provide you with the basic information about the school, the schedule, local arrangements and invited tutorials. Technical information about system installation etc. as well as any updates to the schedule can be found on the project wiki
`https://codex.cs.bham.ac.uk/trac/cogx/wiki/meetings/css11`
which you should be checking regularly in the coming days.
We hope you will all have an interesting learning experience and take home more than just broken arms.

Following the successful format of previous spring schools, this school will also focus on hands-on experience, accompanied by three very interesting invited tutorials.

The topic will be mobile manipulation, i.e. actually fetching that cereal box we have been talking about for some time now. To this end you will mostly be learning about the new manipulation subarchitecture. You will also learn about training and recognising objects in 3D. To safely develp and debug without endangering the precious Katana arms, you will learn how to use the 3D robot simulator. Finally you will encounter a lot of various poses of something w.r.t. something other, which requires proper system calibration.

There will be 2 competitions around the task of fetching cereal boxes, details will be discussed in the respective session.

2.2 Schedule

Wednesday, 27.4.

- arrivals
- 19:00 *Dinner (Wieden Bräu)*

Thursday, 28.4.

- 9:00 - 9:30 Welcome and opening
- 9:30 - 10:45 Tutorial Justus Piater and Renaud Detry
- 10:45 - 11:00 *Coffee*
- 11:00 - 12:30 Tutorial Justus Piater and Renaud Detry
- 12:30 - 13:30 *Catered lunch*
- 13:30 - 14:45 Tutorial Fiona McNeill
- 14:45 - 15:00 *Coffee*
- 15:00 - 16:30 Tutorial Fiona McNeill
- 17:00 - 18:00 **1st (fun) competition**
- 19:00 *Dinner (Wiener Rathauskeller)*

Friday, 29.4.

- 9:30 - 10:45 Tutorial Horst Bischof
- 10:45 - 11:00 *Coffee*
- 11:00 - 12:30 Tutorial Horst Bischof
- 12:30 - 14:00 *Lunch*
- 14:00 - 14:30 Task Description
- 14:30 - 15:30 Technical tutorial Vision and Calibration (Michael)
- 15:30 - 15:45 *Coffee*
- 15:45 - 16:45 Technical tutorial Manipulation, Navigation (Torben)
- 17:00 - 18:30 Robot setup
- 19:00 - 20:30 *Dinner*

Saturday, 30.4.

- 9:30 - 10:00 Morning Q&A
- Hacking
- 12:30 - 13:30 *Lunch*
- Hacking

- 15:00 Social event and Dinner

Sunday, 1.5.

- 9:30 - 10:00 Morning Q&A
- Hacking
- 12:30 - 13:30 *Lunch*
- 16:00 - 16:30 *Coffee and snacks*
- 16:30 - 18:30 **2nd competition**
- 19:00 - 20:30 *Dinner*

Monday, 2.5.

- 9:30 - 10:00 Morning Q&A
- Hacking
- 12:30 - 13:30 *Lunch*
- Hacking
- 16:00 - 16:30 *Coffee and snacks*
- Hacking
- 19:00 - 20:30 *Dinner*

Tuesday, 3.5.

- 9:30 - 10:00 Morning Q&A
- Hacking
- 12:30 - 13:30 *Lunch*
- Hacking
- 16:00 - 16:30 *Coffee and snacks*
- Hacking
- 19:00 - 20:30 *Dinner*

Wednesday, 4.5.

- 9:00 - 9:30 Preparation
- 9:30 - 12:30 **Final Competition**
- 12:30 - 13:30 *Lunch*
- 14:30 - 15:00 Award ceremony and closing
- 15:00 End and safe trips home

2.3 Participants

ALU-FR

- Moritz Goebelbecker (27.4. - 4.5)

BHAM

- Jeremy Wyatt (27.4. - 4.5)
- Charles Gretton (27.4. - 4.5)
- Marc Hanheide (27.4. - 4.5)
- Rustam Stolkin (27.4. - 4.5)
- Marek Kopicki (27.4. - 4.5)
- Claudio Zito (27.4. - 4.5)
- Torben Töniges (27.4. - 4.5)

DFKI

- Geert-Jan Kruijff (27.4. - 4.5)
- Hendrik Zender (27.4. - 4.5)
- Miroslav Janicek (27.4. - 4.5)

KTH

- Alper Aydemir (27.4. - 4.5)
- Yasemin Bekiroglu (27.4. - 4.5)
- Renaud Detry (27.4. - 4.5)
- Kristoffer Sjöö (27.4. - 4.5)

TUW

- Thomas Mörwald (27.4. - 4.5)
- Andreas Richtsfeld (27.4. - 4.5)
- Kai Zhou (27.4. - 4.5)
- Markus Vincze (27.4. - 4.5)
- Michael Zillich (27.4. - 4.5)

UL

- Alen Vrecko (27.4. - 4.5)
- Peter Ursic (27.4. - 4.5)
- Barry Ridge (27.4. - 4.5)
- Danijel Skocaj (27.4. - 29.4)
- Ales Leonardis (27.4. - 29.4)

Invited Speakers

- Fiona McNeill (27.4. - 28.4)
- Justus Piater (27.4. - 30.4)
- Horst Bischof (29.4. - 29.4)

2.4 Teams

Team Red

- Barry Ridge
- Andreas Richtsfeld
- Alper Aydemir
- Rustam Stolkin
- Miroslav Janicek

Team Green

- Thomas Mörwald
- Moritz Göbelbecker
- Jeremy Wyatt
- Alen Vrecko
- Claudio Zito

Team Blue

- Renaud Detry
- Charles Gretton
- Kristoffer Sjöö
- Peter Ursic
- Geert-Jan Kruijff

Team Yellow

- Marek Kopicki
- Hendrik Zender
- Kai Zhou
- Yasemin Bekiroglu

2.5 Local Arrangements

Skipped.

2.6 Invited Tutorials

2.6.1 Learning to Grasp With Grasp Densities

Justus Piater: Intelligent and Interactive Systems Institute of Computer Science University of Innsbruck, Austria
Renaud Detry: Computer Vision and Active Perception lab (CVAP) Kungliga Tekniska Högskolan (KTH), Stockholm, Sweden

Thu, 28.4., 9:30 - 12:30, with 15 min coffee break 10:45

Abstract: We begin with an overview of a selection of important approaches to robotic grasping, including traditional hard-wired approaches, and more recent adaptive methods. We will then present our own work that aims to provide a complete, continuous characterization of the grasp affordances associated with a given object with respect to a specific type of gripper. The idea is to construct, by means of trial grasps, a distribution of object-relative gripper poses, which we call a "grasp density". We discuss generative and discriminative ways of constructing grasp densities, and give introductions to kernel density estimation and regression as they arise in these methods. Finally, we discuss an implementation of these techniques, enabling the participants to employ them in their own work.

2.6.2 Managing Ontologies: Matching and Repair

Fiona McNeill: DReaM group, Centre for Intelligent Systems and their Applications School of Informatics, University of Edinburgh, UK

Thu, 28.4., 13:30 - 16:30, with 15 min coffee break 14:45

Abstract: Using ontologies to represent knowledge is an essential part of interacting with a virtual or real world. But developing usable ontologies is a difficult affair, and inaccuracies and incompatibilities between them and the world are inevitable and can lead to failure and confusion. In this tutorial, I will introduce different kinds of ontologies, explain how the problem of mismatch - between ontologies, and between ontologies and the world - arises, and discuss why it is neither possible nor desirable to remove the possibility of mismatch. I will discuss how such mismatches can be identified, and introduce different approaches to dealing with them, so that successful behaviour becomes possible despite the failure of an ontology to accurately reflect the world.

2.6.3 On-line learning for computer vision

Horst Bischof: Institute for Computer Graphics and Vision TU Graz, Austria

Fr, 29.4., 9:30 - 12:30, with 15 min coffee break 10:45

Abstract: The marriage between computer vision research and machine learning has been very successful and enabled considerable progress in the last few years. In this tutorial I will argue that methods that can learn from data as it arrives (on-line learning) are required to face the next computer vision challenges. Especially robotics is such a domain. I will show the advantages of on-line learning but discuss also the challenges that we have to face. This will lead to semi-supervised learning and multiple-instance learning. Throughout the talk object tracking and object detection will be used as illustrative examples.

Task 1: Sweet Stacks

- Stack Manner Wafers in Gazebo simulator
- The more you stack, the more you score
- there has to be a stack, a single object is not a stack
- Ranking in 1st competition defines starting order for 2nd competition

Scoring

- "flat" stack: 1 point for each wafer on top of another, a single stack of n flat wafers will give n points
- "high" stack: 5 points for each upright wafer being part of a stack, a single stack of n upright wafers will give 5*n points
- wafers in "unstable" positions: 20 points for each wafer in a position, which on its own would be unstable, e.g. one wafer leaning against another
- stacked Wafers must not touch anything besides their supporting Wafer
- minus 5 if you don't get it running on one of your team's laptops ;-)

Prerequisites

You need running versions of player 3.0.2 and gazebo 0.9.

Running

Either check out the spring school system or get the tar ball attached to this page and unpack it e.g. in `/tmp`, which is the option we are going to explain now.

Open a couple of consoles. First you have to let gazebo know where to find models and textures. Edit the file `~/.gazeborc` (assuming you installed the files under `/tmp`)

```
<?xml version="1.0"?>
<gazeborc>
  <gazeboPath>/usr/local/share/gazebo</gazeboPath>
  <gazeboPath>/tmp/task1-files/gazebo</gazeboPath>
  <ogrePath>/usr/local/lib/OGRE</ogrePath>
</gazeborc>
```

Then build a little helper program to open/close the gripper:

```
cd task1-files/gripper
mkdir BUILD
cd BUILD
cmake ..
make
```

Start gazebo

```
cd task1-files/gazebo
gazebo task1.world
```

Start player

```
cd task1-files/player
player cogx-platform-task1.cfg
```

Start playerv, you can add command line options to immediatly subscribe to interfaces upon startup (saves some clicking)

```
playerv --actarray:0
```

Playerv allows you to subscribe and control all interfaces offered by the robots (position2d, camera, ptz, actarray, ..). Sliding the various knobs that appear after selecting to control an interface will move joints etc. Start gripper, the command line option choses which actarray interface (players notion of a robot arm) to chose. Pressing 'o' and 'c' will open and close the gripper.

```
cd task1-files/gripper
BUILD/gripper 0
```

You will now see a gazebo world with several tables and Manner wafers on them. You can control 2 robots, one possibly acting as a "helping hand" to the other. You will need one `playerv` for each robot.

If you have already checked out the spring school system, the respective files can be found under `instantiations/css2011`, and `tools/gripper`.

No Cheating

Of course you can do anything in a simulated world, by editing the world files, disabling gravity etc. The point of this exercise is not however to learn about the intricate details of gazebo world files (and believe me, you don't want to know ...), but to learn controlling the simulation environment as well a getting some insights into the difficulties of approaching and grasping boxy objects with a 5 DOF arm and two finger gripper.

Known Problems

Clicking on an object will select it (a white bounding box is drawn). Moving the mouse will now control the object instead of the view, probably sending the object arcing off in some direction. Pressing the object again will deselect it, giving control back to the view.

Attachments

- task1-files.tgz (1.7 MB) - added by *michaelz* 3 months ago.

Task 2: Hand me the cereals

Time: Monday 16:30 – 18:30

Given:

- 1 High table
- 2 different (chosen by team) cereal boxes placed on the table by judges (both upright, one in direct graspable position (without need to relocate the robot, one turned approx. 90 degrees)

Task:

- Pick up the two cereal boxes and hand it to the user

scores

achievement	score	definition
object detected (each)	100	the team has to prove the object has been detected, either by showing the memory content or the visualisation of the detector
object touched (each) by the gripper	200	the arm has been manipulated and some part of the gripper touches the object
object lifted off the table (each)	400	there is clear space between the object and the table for at least 5 seconds
object handed to the user (each)	300	the object needs to be moved away from the table, the gripper has to be opened, and a team member has to hold the object in her hand
style	0-400	extra scores possible for neat ways of handing the object, artistic manipulation, and others more

Maximum score (without style points): 2x1000=2000

Rules

- the boxes are clearly separated on the same table, however, the exact position is not known
- the team members provide the two objects, they are placed by the judges
- the table and the robot are positioned by the judges
- the object needs to be grasped autonomously
- handing the object *can* involve interaction, but can also be done based on a defined time interval or similar
- one object will be easy to grasp without the need to move the robot (approx 0.5m away in front of the robot), the second one will require the robot to move around the table
- the teams can repeat the task as often as the want in their assigned time slot of 20 minutes. They can restart any time. The maximum score achieved in all runs is the final score. The teams have to leave the arena at the end of their slot sharp!

schedule

Slot	Team	score	position
16:30-16:50	blue	0	4
17:00-17:20	green	900	1 (alea iacta est)
17:30-17:50	yellow	900	2 (alea iacta est)
18:00-18:20	red	500	3

Task 3: Cleaning the Kitchen

Time: Wednesday, 9:30 – 12:30

Given:

- High and low tables (boxes upright or sideways)
- 5 cereal boxes: bircher, chocos, fruchtemusli, toppas, weetabix (NOTE: not the long thin box of platic bags)
- Known room (the computer lab on 4th floor)
- 3 of the 5 cereal boxes
- Several high tables
- 1 low table
- Position of tables unknown

Task:

- find the 3 cereal boxes, located on high tables

scores

achievement	score	definition
each object found	200	teams have to somehow prove they found the object
each object touched by the gripper	200	any black part of the gripper to touch the object
each object lifted off the table	400	the object has to be clearly lifted off the table for at least 5 seconds
each object dropped at the home position (coordinates 0,0; or place with id 0)	300	the robot has to be at the home positions and the gripper released the objects
each object arrived at the low table	400	the object has to be above or max. 50cm away from the drop-off table
each object dropped *on* the low table	300	the points are only scored if the object is *on* the table without making any contact with the robot
each collision of the robot with a table or wall	-50	should any part of the robot touch the table (be it arm, base, gripper, whatever)
emergency stop	-50	should an emergency stop be necessary
style	0-500	extra scores possible for neat ways of handing the object, artistic manipulation, and others more

Maximum score (without style points): 3x1500=4500

Rules

- the arena will be enclosed by artificial walls, the arena is approx. 5x5m in size
- there will be between three and six high tables
- boxes are clearly visible and never hidden
- there will be one or no object on each high table
- an object that drops to the ground is removed from the scene by a judge and put back to its original location immediately.
- the judges will pick three out of the six objects and place them
- there will be one low table (drop-off table) the objects have to be delivered to
- alternatively, objects can be delivered to the home position (coordinates 0,0; or place with id 0) instead of the low table
- the starting position can be chosen by the team (only limitation is, that it has to be more than 1,5 meter away to each object)
- the objects needs to be grasped autonomously
- the teams can repeat the task as often as the want in their assigned time slot (30 minutes). They can

restart any time. The maximum score achieved in all runs is the final score. The teams have to leave the arena at the end of their slot sharp!

- cereals are in any pose, not necessarily upright, but ensured to be graspable,
- the cereals have to be delivered to the low drop-off table using the gripper of the robot
- team members must not touch any computers, robot parts, objects, or tables during the performance (exception: the safety person) . If a team members touches anything like this, the current run will be canceled and a restart will be necessary
- nobody besides the safety person and a judge must be inside the arena
- an example setup is arranged in the arena now, the actual setup for the competition will qualitatively (orientations, relations, etc.) similar, but will differ in exact positions and selected objects.
- teams are allowed to practice in the arena any time until 4/5/2011 9:30am

schedule

Slot	Team
09:45-10:15	~~blue~~ more sleep and setup time instead
10:30-11:00	~~red~~ (red unfortunately resigned) **blue**
11:15-11:45	**yellow**
12:00-12:30	**green**

points

Position	**Team**	Found Objects	Touched Objects	Lifted Objects	Dropped Objects	Collision	Emergency-stop	style	**all**
1	**green**	3 -> 600 points	2 -> 400 points	2 -> 800 points	2 -> 600 points			200 points (pan-tilt object search + smart strategy)	**2600 points**
2	**blue**	2 -> 400 points	1 -> 200 points	1 -> 400 points	1 -> 300 points			200 points (robustness + smart strategy)	**1500 points**
3	**yellow**	1 -> 200 points						100 points (speech usage)	**300 points**
4	**red**								**0 points**

CogX Spring School Vienna 2011

Task Description
Technical Tutorial

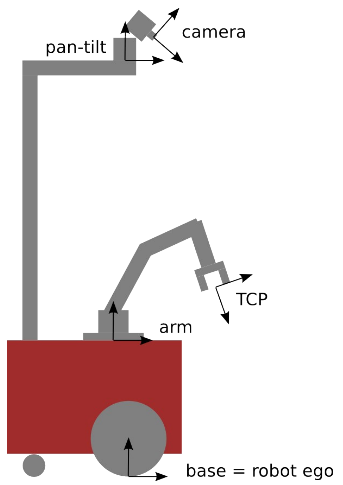
Overall

Cleaning up after breakfast

The whole family had a healthy breakfast and everybody left her/his cereal box somewhere in the kitchen.

Mani, the robot should find all the cereal boxes standing on various tables and deliver them to a drop off place.

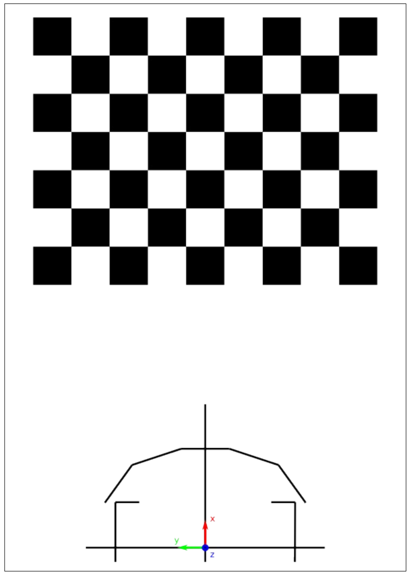
Coordinate Systems

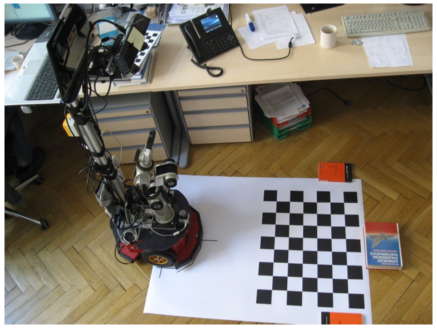


Calibration

- Left camera, intrinsic (camcalib)
- Right camera, intrinsic (camcalib)
- Stereo system, intrinsic (SVS)
- Kinect, intrinsic (camcalib)
- Left camera, pose in robot ego (ptucalib_simple)
- Right camera, pose in robot ego (ptucalib_simple)
- Kinect camera, pose in robot ego (ptucalib_simple)
- Arm, pose in robot ego (manual)

Calibration: Camera Poses





Math Tools

tools/math/src/c++/math

- Box3.h
- cogxmath_base.h
- cogxmath.h
- Matrix33.h
- Plane3.h
- Pose3.h
- Sphere3.h
- Vector2.h
- Vector3.h

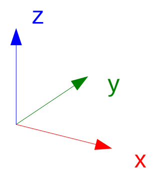
Examples (see header)

```
/** Transform 3D point from local to world coordinates, pose T = [R, p].
 * b = R * a + p;  */
inline Vector3 transform(const Pose3 &T, const Vector3& a)

/** Transform 3D point from world to local coordinates, pose T = [R, p].
 * b = R^T * (a - p) */
inline Vector3 transformInverse(const Pose3 &T, const Vector3& a)

/** Transform 3D direction vector from local to world coordinates, pose T =
[R, p].
* b = R * a; */
inline Vector3 transformDirection(const Pose3 &T, const Vector3& a)

/** Transform pose from local to world coordinates, poses T = [R, p], T1 =
[R1, p1], T2 = [R2, p2].

 * T2 = [R * R1, R * p1 + p] */

inline void transform(const Pose3 &T, const Pose3& T1, Pose3& T2)
```

Handy Stuff

- Notation:
  Transformation from **C**amera to **W**orld: ${}^{W}T_{C}$
- Thus ${}^{G}T_{C} = {}^{G}T_{W}\,{}^{W}T_{C}$
- Row vectors of rotation matrix are axis vectors of camera system in world coords:

$$\begin{vmatrix} * & * & * \\ * & * & * \\ * & * & * \end{vmatrix}$$



Katana First Aid

Jörn Kusel

mechatronic

Burghalde 10

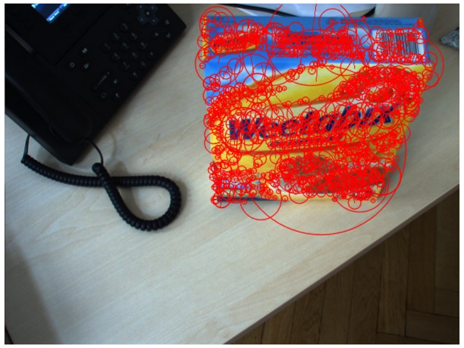
9100 Herisau

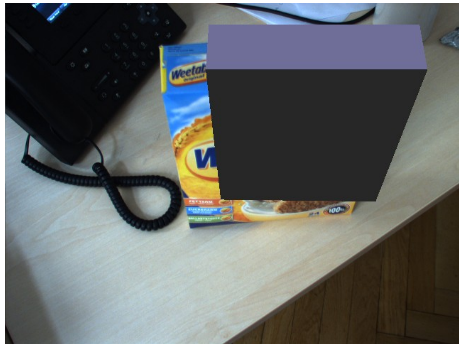
Switzerland

Phone: 0041 (0)79 450 9486

Learning Objects

- learn-object.cast





<space> to learn view
<s> to save model
NOTE: present models are extended

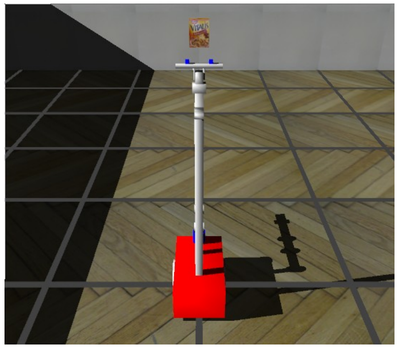
<l> lock/unlock tracking
<t> to learn texture
<s> to save model
<m> to change view mode

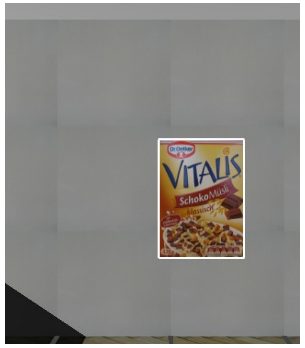
Important command line params

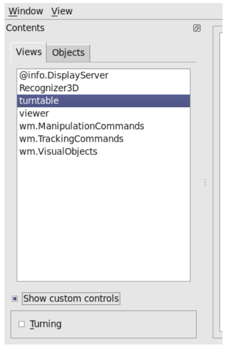
- CPP MG CameraMount CameraMount --camids "0" --cam_poses_xml instantiations/css2011/calibration/ptu-pose-cam-left-sim.xml --pt_base_xml instantiations/css2011/calibration/ptu-pose-base-sim.xml --pt_pan_xml instantiations/css2011/calibration/ptu-pose-pan-sim.xml --pt_tilt_xml instantiations/css2011/calibration/ptu-pose-tilt-sim.xml **--fixed_cam_poses_xml "instantiations/css2011/calibration/example-cam-left-pose.xml" --fixed_pan_tilt "0 -0.785398175"**
- CPP MG Recognizer3D ObjectRecognizer3D --videoname VideoServer --camid "0" --display **--plyfiles "instantiations/css2011/object-models/example-cereals-weetabix.ply" --siftfiles "instantiations/css2011/object-models/example-cereals-weetabix.sift" --labels "example-cereals-weetabix" --initpose "[0.56 0.0 0.85][0.0 0.0 1.57]"** --log

Learning Objects in Simulation

- gazebo learn.world
- learn-object-sim.cast







@info.DisplayServer
Recognizer3D
turntable
viewer
wm.ManipulationCommands
wm.TrackingCommands
wm.VisualObjects

Show custom controls

Turning

Materials for the Spring School 2011

Most important notes

- **stick to your team's svn, don't mess around, don't commit changes to any externals (`tools/*`)**: The teams' svns are located under systems/css-2011-teams, the svn URLs are:
  - `https://codex.cs.bham.ac.uk/svn/nah/cogx/code/systems/css-2011-teams/red`
  - `https://codex.cs.bham.ac.uk/svn/nah/cogx/code/systems/css-2011-teams/green`
  - `https://codex.cs.bham.ac.uk/svn/nah/cogx/code/systems/css-2011-teams/blue`
  - `https://codex.cs.bham.ac.uk/svn/nah/cogx/code/systems/css-2011-teams/yellow`
- **Make sure there is always one person ready to cut the power to the arm when operating the real robot.** Be aware that the arm just drops down and needs support as soon as power is cut off.

Preparations

- dependencies setup with the GAR installer
- cogx system installation
- system setup and preparation
- There are further tutorials that could be useful for specific components:
  - Install OpenCV
  - Install Kinect

Tasks

- Task 1: Sweet Stacks
- Task 2: Hand me the Cereals
- Task 3: Cleaning the Kitchen

Relevant Tutorials

- How to calibrate the various parts of the system: System Calibration
- How to calibrate the cameras: Camera Calibration?
- How to communicate with the arm: manipulation.sa Tutorial; also very useful: the source code of the GUI tool to understand how commands are submitted to working memories.
- How to detect objects: Object Recognizer 3D
- How to access the local maps: LocalMapManager Tutorial
- How to move the robot:
  - these tutorials are horribly outdated; don't follow them, but only use them as a reference: Nav Tutorial from first spring school, also see the accompanying slides
  - it's to better to check systems/spring-school-2011/subarchitectures/spatial.sa/src/java/spatial/manual/ManualNavGUI.java to understand how to send the robot around.
  - look at the bottom of basicManipulationSim.cast where you can see visualisation component registered on all the relevant types you need for moving the robot and reading the position:
    - `SpatialData.NavCommand` to send the robot around (also see GUI source)
    - `NavData.RobotPos2d` is continuously updated with the current pose of the robot in the global (map) coordinate system
    - `SpatialData.Place` is the data type for places and place holders as the are also displayed in peekabot
    - **be aware that you have to submit the `NavCommand` commands to the spatial working memory!!!**
- How to access the pan-tilt-unit: Pan-Tilt hardware access; again, the source code is probably the best example: tools/hardware/ptz/branches/spring-school-2011/src/java/ptz/ptzServer

/PanTiltZoomServer.java

- How to work with gazebo worlds and models: Gazebo worlds and models
- How to work with plane pop-out to detect tables: Plane Pop-Out
- Further tutorials are available here. some other components we use are documented here:
    - Video Servers?
    - Video Clients?
    - Point Cloud Servers?
    - Point Cloud Client?
    - Camera Mount
    - Point Cloud Viewer
    - TomGine
    - Gazebo Vision
    - Visualization SA
    - CAST Control

best practices

- use logging:
    - you can configure logging in castctrl, enable logging to a file as well
    - use the nice log browser logmx, installed in `tools/logtools/LogMX/logmx.sh`
- start from the example cast file systems/spring-school-2011/instantiations/css2011/basicManipulationSim.cast to create your own system
- use the Working Memory viewer components to inspect the working memory contents in the viewer (see end of example cast file).
- use CAST Control to start everything, see video for the settings.
    - start it using `./castcontrol.py`
- check the tutorial video to understand the basic system
- ask your group fellows first, and ask Torben, Michi, or Marc should you have problems second ;-)

Tips & Tricks

- there is a new set of static methods available in java to transform poses. check it out at tools/math/src/java/mathlib/Functions.java.
- Player sometimes causes problems when started in castctrl; it is useful to start it separately via `player instantiations/css2011/player/cogx-platform-sim.cfg`
- Alternatively: use a cleanup-script in castcontrol which contains the following line:

```
rm /tmp/gazebo-$USER-0 -rf
```

- if you have strange crashes (with nothing showing in peekabot) make sure `~/.peekabot/data` is a symbolic link to your instance of `instantiations/peekabot-models/data`; if not, you have to create it with `ln -s`
- if you have problems with the installation look also at this page
- gazebo patches 1, gazebo patches 2, gearbox patch, player-3.0.2 patch (Hint: you have to click on the bottom of the page at the link "Plain text" to get the file)
- in Virtualbox:
    - it seems there is a problem with Golem when guest additions are installed. A work-around for this is described here: http://forums.virtualbox.org/viewtopic.php?f=3&t=30964; alternatively, you of course can just *not* install the guest additions
- SiftGPU and CUDA problems: If you have problems with SiftGPU (needed for ObjectRecognizer3D) complaining about missing CUDA stuff, set the following cmake option (also set in the init installation script described here)
    - LIBBUILD_SIFTGPU ON
    - LIBBUILD_SIFTGPU_WITH_CUDA OFF
    - LIBBUILD_CUDASIFT OFF
    - LIBBUILD_PYTHON_VISION OFF
    - LIBBUILD_LEVMAR_HOMEST ON

    - LIBBUILD_LEVMAR_HOMEST_DEMO OFF
- Note that the PTZ GUI outputs and takes **degrees** while component command line parameters are all in **radians**. That is a likely source of errors (and we should have avoided the different units of course).

Attachments

- setup.ogv (0.6 MB) - added by *hanheidm* 3 months ago. “setting for CASTCtrl”
- testrun.ogv (3.5 MB) - added by *hanheidm* 3 months ago. “a test run to see how to control the system”
- css2011-techtutorial.pdf (0.6 MB) - added by *michaelz* 3 months ago.

Using the GAR installer

The basis for this is here: Setting up DORA year2 system

It is assumed, that a java jdk is already installed and configured and that the partner repositories of synaptic are enabled.

1. Create a local folder for the Cogx code -- my local folder for all stuff will be `~/cogx`; e.g., `mkdir cogx; cd cogx`
   1. Get the gar-installer: `svn co https://codex.cs.bham.ac.uk/svn/nah/cogx/code/tools/gar-installer/trunk gar-installer`: We provide a so-called GAR-installer to ease installation.
   2. choose the configuration which represents the setup of your robot and change to that folder
      - `cd gar-installer/cogx/summerSchool_katana300_x32`: x32 system in combination with a katana 300 manipulator
      - `cd gar-installer/cogx/summerSchool_katana300_x64`: x64 system in combination with a katana 300 manipulator
      - `cd gar-installer/cogx/summerSchool_katana450_x32`: x32 system in combination with a katana 450 manipulator
      - `cd gar-installer/cogx/summerSchool_katana450_x64`: x64 system in combination with a katana 450 manipulator
   3. Let the installer fetch and compile all external dependencies: `sudo make install`
      GAR has a management of dependencies. So we can now simply change into the relevant directory and tell GAR to install this. It will automatically also fetch, configure, build, and install all require packages. So, let's do it (here a 'sudo' is required to gain administrative permissions, enter your password when being asked for it.
   4. If you need Mercury or Google Protocol Buffers you have to install it manually according to the instructions listed here.

Known problems

player

On 64-bit systems it can happen that `pkg-config` does not find `player` after installation, e.g. `pkg-config --cflags playerc++` does not work. The reason is that the `pkg-config .pc` files are normally installed under

```
/usr/local/lib/pkgconfig/playerc++.pc
etc.
```

which is OK on a 32 bit system. On a 64 bit system however they end up in

```
/usr/local/lib64/pkgconfig/playerc++.pc
```

There *should* be a symlink from `lib` to `lib64` (as is the case in `/usr/lib`), but that might not be the case. Just copy

```
cp -a /usr/local/lib64/pkgconfig /usr/local/lib/pkgconfig
```

gazebo (Ogre Path)

Note: after installation you have to modify `~/.gazeborc`. Otherwise you will get the following error

```
[/usr/local/src/gazebo-0.9.0/server/GazeboConfig.cc:115]
Ogre Path[OGRE]
Error Loading Gazebo
/usr/local/src/gazebo-0.9.0/server/rendering/OgreAdaptor.cc:428 : Except
```

For some reason `~/.gazeborc` is owned by `root`. So first make it owned by you

```
chown myself.mygroup ~/.gazeborc
```

then change

```
<ogrePath>OGRE</ogrePath>
```

to

```
<ogrePath>/usr/lib/OGRE</ogrePath>
```

gazebo (no textures and models)

To see all used textures and models, you have to adapt the `~/.gazeborc` again.

Therefore you have to add

```
<gazeboPath>/vol/cogx/spring-school-2011/instantiations/css2011/gazebo</
```

as a further gazebo path in the gazeborc.

peekabot installation under ubuntu 10.10

If you get an error like this:

```
libtool: link: cannot find the library `/usr/lib/libgdk_pixbuf-2.0.la' o
make[4]: *** [peekabot] Error 1
make[4]: Leaving directory `/home/ursicp/cogx/systems/springSchool11/gar
make[3]: *** [all] Error 2
make[3]: Leaving directory `/home/ursicp/cogx/systems/springSchool11/gar
make[2]: *** [all-recursive] Error 1
make[2]: Leaving directory `/home/ursicp/cogx/systems/springSchool11/gar
make[1]: *** [build-work/main.d/peekabot-0.8.4/Makefile] Error 2
make[1]: Leaving directory `/home/ursicp/cogx/systems/springSchool11/gar
make: *** [../../contrib/peekabot/cookies/main.d/install] Error 2
```

A solution which worked before is the following:

1. search for all `libgdk_pixbuf-2.0.la` (e.g. by `grep /usr/lib/libgdk_pixbuf-2.0.la /usr/lib/*.la`)
2. change in every founded file the entry `/usr/lib/libgdk_pixbuf-2.0.la` to `-lgdk_pixbuf-2.0`
3. the same principle has to be done for your gtk_pixbuf as well

java jdk not found

If something like this occur:

```
BUILD FAILED
/home/mz/Downloads/cogx/gar-installer/cogx/cast-java/work/main.d/cast-sv
com.sun.tools.javac.Main is not on the classpath.
Perhaps JAVA_HOME does not point to the JDK.
It is currently set to "/usr/lib/jvm/java-6-openjdk/jre"
```

You have to install the JDK to build the java parts: An easy way is:

```
sudo apt-get install openjdk-6-jdk
```

OpenCV under ubuntu 11.04

you might get the error:

```
/home/user/OpenCV-2.2.0/modules/highgui/src/cap_v4l.cpp:217:28: fatal er
compilation terminated.
make[2]: *** [modules/highgui/CMakeFiles/opencv_highgui.dir/src/cap_v4l.
make[1]: *** [modules/highgui/CMakeFiles/opencv_highgui.dir/all] Error 2
make: *** [all] Error 2

or

"/lib/libopencv_highgui.so.2.2.0: undefined reference to `cvCreateCamera
```

you should download this patch: https://code.ros.org/trac/opencv/attachment/ticket/862/OpenCV-2.2-nov4l1.patch

Installation of the cogx system itself

For the installation of the cogx system itself, I adapted Marko's installation for the george year 3 system:

1. Go to your 'local folder', e.g., `cd ~/cogx`
2. Get the spring school sources from svn: `svn co https://codex.cs.bham.ac.uk/svn/nah/cogx/code/systems/spring-school-2011`
3. Change to the downloaded folder: `cd spring-school-2011`
4. Run "`bash sysconfig/prepare_build.sh`" to create an initial `BUILD/CMakeCache.txt` with my current compile settings
    - If you want to restore the set of components that are built in the summer school system, use the following command: `tools/scripts/cmake-apply BUILD sysconfig/cmakecache/summerSchool2011.txt`
    - To save the current set of components: `tools/scripts/cmake-freeze > my-components.txt`
5. Build the system (you can use CAST control or build it manually as describes in the following):
    1. Change to your BUILD folder: `cd BUILD`
    2. Run ccmake: `ccmake ..`
    3. Edit the compile options to adapt your system configuration
    4. Edit the `OUTPUT` option so that it corresponds to a directory that exists, and for which you have write access.
    5. Compile everything: `make -j4 install`. Sometimes you have to repeat this step a couple of times because of the "-j 4" (i.e., threaded compilation) option tripping over itself.

Required external software

If you do not use the gar-installer, you have to install the following software manually. Also have a look at the Known Problems section on http://codex.cs.bham.ac.uk/trac/cogx/wiki/meetings/css11/material/setupEasy.

peekabot-0.8.4

http://sourceforge.net/projects/peekabot/files/peekabot/0.8.x/peekabot-0.8.4.tar.bz2/download

PhysX

https://codex.cs.bham.ac.uk/svn/nah/cogx/code/tools/gar-installer/trunk/contrib/physx_x32/files/PhysX_x32-2.8.3.3.tar.gz

or

https://codex.cs.bham.ac.uk/svn/nah/cogx/code/tools/gar-installer/trunk/contrib/physx_x64/files/PhysX_x64-2.8.3.3.tar.gz

(contains deb files to install)

Golem

https://codex.cs.bham.ac.uk/svn/nah/cogx/code/tools/gar-installer/trunk/cogx/golem_katana300_x32/files/golem-trunk.tar.gz

(The tar.gz file contains all relevant files for both arms, x32 and x64 systems. It can be configured with cmake)

You can configure whether you want to build 32 or 64 bit:

```
BUILD_32_BIT                      ON/OFF
```

You can configure whether you build Katana 450, which includes the AXNI interface:

```
BUILD_DEVICE_KATANA450            ON
BUILD_AXNI                        ON
```

The Katana 450 AXNI interface requires the pcan CAN bus driver:

https://codex.cs.bham.ac.uk/svn/nah/cogx/code/tools/gar-installer/trunk/contrib/peak-linux/files/peak-linux-driver.6.20.tar.gz

player 3.0.2

http://sourceforge.net/projects/playerstage/files/Player/3.0.2/player-3.0.2.tar.gz/download

gazebo 0.9

0.9 http://sourceforge.net/projects/playerstage/files/Gazebo/0.9.0/gazebo-0.9.0.tar.bz2/download

Note: after manual installation (without the gar installer) you have to provide the following symlink:

```
cd /usr/local/include/player-3.0:
sudo ln -s libplayerinterface libplayerxdr
```

Note: after installation you have to modify `~/.gazeborc`. Otherwise you will get the following error

```
[/usr/local/src/gazebo-0.9.0/server/GazeboConfig.cc:115]
  Ogre Path[OGRE]
Error Loading Gazebo
/usr/local/src/gazebo-0.9.0/server/rendering/OgreAdaptor.cc:428 : Except
```

For some reason `~/.gazeborc` is owned by `root`. So first make it owned by you

```
chown myself.mygroup ~/.gazeborc
```

then change

```
<ogrePath>OGRE</ogrePath>
```

to

```
<ogrePath>/usr/lib/OGRE</ogrePath>
```

openCV 2.2

http://downloads.sourceforge.net/project/opencvlibrary/opencv-unix/2.2/

System Calibration

Various parts of the system need to be calibrated, so that everything can be treated as relative to the same robot ego pose.

- Left camera, intrinsic parameters, using camcalib?
- Right camera, intrinsic parameters, using camcalib?
- Kinect, intrinsic parameters, using camcalib?
- Stereo system, intrinsic parameters, using SVS smallvcal? (Small Vision System by Videre design) **NOTE**: the calibration pattern we use at the spring school has a square side length of 28 mm, which has to be entered as a custom setting in smallvcal.
- Left camera, pose in robot ego, using ptucalib_simple?
- Right camera, pose in robot ego, using ptucalib_simple?
- Kinect camera, pose in robot ego using ptucalib_simple?
- Arm pose in robot ego, manually measured

These are used by components

- video server
- camera mount
- manipulation server

Video Server

The left, right and kinect intrinsic parameters, named e.g. `camcalib-left.xml, camcalib-right.xml and camcalib-kinect.xml` or `cam-left-cal.xml etc.` are entered as command line parameters of the video server, e.g. from `instantiations/css2011/basicManipulationSim.cast`

```
CPP MG VideoServer PlayerVideoServer --camids 0 --devnums 0 --imgsize "64
```

Camera Mount

The left, right and kinect poses in robot ego, named e.g. `campose-left.xml, campose-right.xml, campose-kinect.xml` or `example-cam-left-pose.xml, etc.` are entered as command line parameters for the camera mount component, e.g. in `instantiations/css2011/check-system-calibration`

```
CPP MG CameraMount CameraMount --camids "0" --fixed_cam_poses_xml "instan
 --fixed_pan_tilt "0 -0.785398175" --cam_poses_xml instantiations/css201
 --pt_base_xml instantiations/css2011/calibration/ptu-pose-base-sim.xml
 --pt_pan_xml instantiations/css2011/calibration/ptu-pose-pan-sim.xml
 --pt_tilt_xml instantiations/css2011/calibration/ptu-pose-tilt-sim.xml
```

I.e. they are entered as the `--fixed_cam_poses_xml`, which were calibrated with the pan/tilt angles at `--fixed_pan_tilt`. The other poses provided to `CameraMount` are kinematic parameters of the PTZ, and were measured by hand.

Manipulation Server

The arm pose has to be measured by hand for the time being (a proper calibration procedure is being developed), by manually measuring the position of the center of the arm base to the robot origin, which lies precisely between the wheels, with x pointing forward, and y pointing to the left. This pose is added as command line parameter to the manipulation server

```
JAVA MG manipulationServer manipulation.runner.cogx.CogXRunner --configPa
```

And an example can be found in `instantiations/css2011/calibration/example-arm-pose.xml`

Checking System Calibration

You can check that system calibration is correct, i.e. all poses were correctly specified as various command line parameters, by running the `CheckSystemCalibration` component, e.g. `instantiations/css2011/check-system-calibration.cast`

```
CPP MG checksystemcalibration CheckSystemCalibration --videoname VideoSe
```

It offers a check button in to CAST viewer GUI to start/stop streaming images. While it streams images, it will display the projected calibration pattern of the calibration poster. If you put the robot on the calibration poster and move the pan/tilt down, you should see the projected calibration pattern overlaid over the actual calibration pattern. The images are not undistorted, so a few pixels off (depending on lens distortion) are ok. Moving the pan/tilt sideways and up/down the projected calibration pattern should stay on top of the actual calibration pattern. If that is the case, then all camera and pan/tilt related poses are correct. See attached images at the end of this page, for e.g. the left camera and kinect.

Attachments

- check-sys-calib-L.jpg (36.1 KB) - added by *michaelz* 3 months ago.
- check-sys-calib-K.jpg (45.4 KB) - added by *michaelz* 3 months ago.

The manipulation subarchitecture

Definiton of Arm Coordinate Systems

Note that the definition of the arm coordinate system follows a different scheme than that of the robot base.

The arm coordinate system is defined as indicated below. I.e. if the arm is fully outstretched the tool center point (TCP) pose will be such that its x axis points to the right (in -y direction of the robot ego coordinate system), its y axis points forward (in +x direction of the robot ego coordinate system) and z points upward (in +z direction of the robot ego coordinate system).

The rotation of the arm base w.r.t. robot ego is taken into account in the arm pose, e.g. `instantiations/css2011/calibration/example-arm-pose.xml`.

```
            t1    t2               t3              t4
            ^    ^                ^          ^ Z  ^
            |   /                /           |   /
            |  /                /            |  /
            | /                /             | /
            |/       l1       /      l2       |/      Y
            0****************0***************0=========> t5
           /*               /               /
          / *              /               /   T
         /  *             /               /
        /   * l0          /              V X
            *
            *^ Z
            *|
            *|
            *| S     Y
             0========>
            /
           /
          / X
         v
```

If in doubt about coordinate axes use the get postition command of the manipulation GUI and look at the column vectors of the rotation matrix, which denote the x, y, and z axes of the TCP coordinate system.

Prerequisites

The TinyIce-Interface of GOLEM is needed to communicate with the attached manipulator.

To quickly test whether Golem and the arm work, you can use one of the many demo programs provided with Golem. The following will move a simulated Katana to its outstretched (zero) position:

```
cd GOLEM_INSTALLATION_DIRECTORY
cd resources/Device
../../bin/linux/PhysJoint ../Demo/PhysJoint.xml
```

(Note that pathnames and locations do matter!)

Now change in file `../Demo/Joint.xml` `GolemDeviceKatana300Sim` to `GolemDeviceKatana450`. Plug in the arm, **KEEP YOUR HAND ON THE EMERGENCY SHUTDOWN**, and run

```
../../bin/linux/Joint ../Demo/Joint.xml
```

The real arm will calibrate, and then move to the same outstretched position **WITHOUT CHECKING FOR COLLISIONS**! So depending on how and where you mounted your arm, that movment might or **MIGHT NOT BE COLLISION FREE**. Cut power immediately if you think the arm is going to collide. Especially watch for cables, dangling from the robot superstructure if the arm is mounted on the robot.

If this worked, your arm is ready to be used with Golem inside the CAST system.

Known Problems

The Katana300 uses a serial port. Golem's Katana300 driver assumes that to be `/dev/ttySX` with `X = 0, 1, ...`. The port number is configured via file `GolemDeviceKatana300.xml`, first line:

```
<arm name="Katana 300 (6M180)" custom_kinematics="0" cfg_path="katana300_
```

When using a USB to serial adapter you are likely to get COM ports such as `/dev/ttyUSB0`. You will have to create a symbolic link

```
sudo ln -sf /dev/ttyUSB0 /dev/ttyS1
```

to make Golem use that port.

Manipulation Server

The component `manipulationServer` is needed to control the arm via the CAST working memory.

The command line parameter are listed below:

`--configPath <string>` ... defines the path to the calibration file

`--armName <string>` ... specifies the attached arm. Possible values are: "`simulation`" to use the simulation environment, "`katana300`" to use the katana 300 manipulator and "`katana450`" to use the katana 450 manipulator

`--testGUI` ... enables a simple GUI to test the commands

An example of the whole sequence:

```
JAVA MG manipulationServer manipulation.runner.cogx.CogXRunner --configPa
```

used enumerations

`ManipulationCompletion`

- `FAILED` ... execution of the command failed (is set by the system)
- `SUCCEEDED` ... command was successful (is set by the system)
- `ONTHEWAY` ... manipulator is moving (is set by the system)
- `COMPINIT` ... initial value (can be set by the user)

`ManipulationCommandStatus`

- `NEW` ... the command is new (can be set by the user)
- `CHANGED` ... the command was already in the memory and changed (can be set by the user)
- `PENDING` ... the system is currently executing the command (is set by the system)
- `FINISHED` ... the command is finished (is set by the system)
- `COMMANDFAILED` ... the command failed (is set by the system)

`GraspingStatus`

- `GRASPINGSTATUSINIT` ... initial value (can be set by the user)
- `GRASPING` ... the gripper is currently grasping an object / gripper is not fully closed (is set by the system)
- `NOTGRASPING` ... the gripper is not grasping an object / gripper is fully closed (is set by the system)

available commands

If one of the following commands is added to the working memory, the PTZ-Unit will execute the corresponding action.

`ManipulationCommand`

This class represents an overall class of all manipulation commands.

Fields:

- `ManipulationCommandStatus status` ... current status of the command
- `ManipulationCompletion comp` ... current completion of the command

It has to be noticed, that while GOLEM calculates a solution, there is no way to interrupt the calculation itself. Only if the `comp` field changes to `ONTHEWAY` an interuption is possible.

`ManipulationExternalCommand (ManipulationCommand)`

This class represents an overall class of all external commands. These commands can be used by the user.

`ManipulationInternalCommand (ManipulationCommand)`

This class represents an overall class of all internal commands. The system itself is using these commands to communicate. The user should not use internal command.

`PutDownCommand (ManipulationExternalCommand)`

This command implements a simple put down action. It tries to put a given object on a visual object which WM address is known. It has to be noticed, that the arm movement planning software return a valid solution for every visual object most of the time. Furthermore no rotation of the based object are taken into account.

The command itself will not check, if the visual object is in the range of the manipulator. It will move to the nearest possible position to the visual object and open its gripper 5 cm over the midpoint of the object.

Field:

- `cast::cdl::WorkingMemoryAddress basedObjectAddr` ... address of the visual object the current grasped object should be placed on

`FarArmMovementCommand (ManipulationExternalCommand)`

This command represents a simple far arm movement. It tries to reach the midpoint of the object with a fixed gripper orientation (parallel to the floor)

Fields:

- `cast::cdl::WorkingMemoryAddress targetObjectAddr` ... address of the visual object to approach
- `cogx::Math::Pose3 reachedPose` position, the manipulator has reached (set by the system)

`FineArmMovementCommand (ManipulationExternalCommand)`

This command represents a simple grasp approach arm movement. The manipulator will move from the current position to the midpoint of the object and the gripper will be closed. The orientation will be the same orientation like the orientation of the movement starting point of this command.

Fields:

- `cast::cdl::WorkingMemoryAddress targetObjectAddr` ...address of the visual object to approach
- `GraspingStatus graspStatus` ... grasped an object or not (set by the system)

`SimulateFarArmMovementCommand (ManipulationExternalCommand)`

This command simulates the movement described in the `FarArmMovmentCommand`.

Fields:

- `cast::cdl::WorkingMemoryAddress targetObjectAddr` ... address of the visual object to approach
- `cogx::Math::Pose3 simulatedReachablePose` ... represents the pose of the manipulator which would be reached (set by the system)

`StopCommand (ManipulationExternalCommand)`

This command stops the arm movement

`MoveArmToHomePositionCommand (ManipulationExternalCommand)`

This command moves the arm to its home position (initial position of the arm after the calibration).

`OpenGripperCommand (ManipulationExternalCommand)`

This command opens the gripper of the manipulator.

`CloseGripperCommand (ManipulationExternalCommand)`

This command closes the gripper of the manipulator.

`MoveArmToPose (ManipulationExternalCommand)`

This command moves the manipulator to a pose (position and rotation) defined by the field targetPose.

`SimulateMoveToPose (ManipulationExternalCommand)`

This command simulates the `MoveArmToPose` command.

Fields:

- `cogx::Math::Pose3 targetPose` ... target pose to simulate to
- `cogx::Math::Pose3 simulatedReachablePose` ... represents the pose of the manipulator which would be reached (set by the system)

`GetCurrentArmPose (ManipulationExternalCommand)`

This command returns the current pose of the manipulator.

Field:

- `cogx::Math::Pose3 currentPose` ... current pose of the manipulator (set by the system)

PlayerArmBridge

Provides a bridge between Golem path planning and a player actarray interface. This is needed to have Golem control an arm simulated in gazebo.

Command line parameters are:

`--playerhost <string>` ... hostname of player server, default `localhost`

`--playerport <int>` ... player port, default `6665`

GazeboTurntable

Uses the player/gazebo simulation interface to continuously rotate a given object. Useful for learning virtual objects.

Command line parameters are:

`--playerhost <string>` ... hostname of player server, default `localhost`

`--playerport <int>` ... player port, default `6665`

`--label <string>` ... the label of the object that should be rotated

Object Recognizer 3D

This component can recognize 3D object instances defined as surface geometry plus SIFT features attached to these surface.

It will output `VisualObjects` defined in `Vision.ice` in the `vision.sa`. The most relevant fields are:

```
class VisualObject {
  // 3D position and orientation, in the robot ego coordinate system.
  cogx::Math::Pose3 pose;

  // The time when the object was last observed, in any view
  cast::cdl::CASTTime time;

  // Geometric representation in 3D space
  GeometryModel model;

  // Distribution of identity labels (from object recognizer)
  StringSeq identLabels;
  DoubleSeq identDistrib;
  double identGain;
  double identAmbiguity;
  ...
};
```

As cou can see the object label is given as a probability distribution over labels, e.g. ("cereals-weetabix", "unknown") with probabilities (0.3, 0.7). The probabilites are given by the detection confidence of the recognizer and are thus not true probabilites. Note that confidences (probabilities) around 0.3 are considered good recognition results. There is always an "unknown" label which makes probabilities sum to 1.

ObjectRecognizer listens to the following commands from `VisionData.ice`

```
class DetectionCommand {
  StringSeq labels;
};

enum Recognizer3DCommandType{ RECSTOP, RECLEARN, RECOGNIZE };
class Recognizer3DCommand{
      Recognizer3DCommandType cmd;
      string label;
      string visualObjectID;
      double confidence;
};
```

The `DetectionCommand` is the one you want to use. Simply provide the labels of the objects you want to recognize and the recognizer will try to find them and put `VisualObjects` on the working memory (WM). Of course the recognizer must have SIFT models for these labels and be configured to load them in its CAST file command line. The `Recognizer3DCommand` command is used internally between the `ObjectTracker` and `ObjectRecognizer3D` during learning.

(When asked to recognize an object via a `Recognizer3DCommand` (containing object label and command type RECOGNIZE) it will place a `VisualObject` with label, 6D pose and detection confidence in visual WM. If command type is RECLEARN, the recognizer will learn the given model. To this end it will require a geometry model file (in PLY format, e.g. created using blender). It will issue commands to the tracker to track this model and add SIFT features to the surfaces while the user presents new views of the object. The model will be "frozen" in the tracker window at first, at an arbitrary but reasonable pose. Align the physical object with its frozen model (roughly) and press 'l' in the tracker window to unlock the model, it will now be tracked as you move it around. Go to the recognizer window and press ' ' (space) whenever you want to learn a new set of SIFT feature for the current view. Learn a new view roughly every 30 degrees. Press 's' in the recognizer window to save the learned SIFT model or 'q' to cancel learning.)

Note that `ObjectRecognizer3D` will always put a visual object in WM, with a 3D pose and a label distribution.

- If an object could be found: probability of its label is between (0, 1]
- If an object could not be found: probability of its label is 0, pose is identity
- Upon first calling the recognizer, it will add an object
- Calling it again with the same object will overwrite the object already in WM.
- The recognizer will not delete or duplicate objects. It is an object instance recognizer, i.e. objects are considered unique (hence no duplication). Furthermore it does not keep track of detected objects, so can not tell when they are no longer visible (hence no deletion).

So as a caller of `ObjectRecognizer3D` you issue a `DetectionCommand` and wait for ADD or OVERWRITE of `VisualObjects`.

The label probability value is essentially derived from the percentage of SIFT features that could be matched. From experience values below 0.08 can be considered to be false positives, values above 0.2 can be considered quite good results. But note that this value depends on a several factors and behaves differently for different objects, depending on the amount and type of texture, object size etc.

There is an example cast file in `subarchitectures/vision.sa/config/test-recognizer3D/single-recognizer3D.cast` For the spring school, `instantiations/css2011/learn-object.cast` and `instantiations/css2011/find-object.cast`.

Keyboard mapping

to press inside the recognizer window

- [ ] (space) .. add current view to model (note: make sure the tracker has currently a good track of the object)
- [s] .. save the current model with its SIFT features and quit learning
- [q] .. cancel learning

CAST GUI

ObjectRecognizer3D also uses the CAST GUI to display recognized objects, so in recognition mode, where you don't need the above keyboard commands, you can (and should) disable the OpenCV window (i.e. not use `--display`). The CAST GUI offers a button per learned object. Pressing that button will recognize the respective object in the current image.

CAST command line parameters

`--videoname <string>` .. component ID of the video server component to connect to

`--camid <int>` .. ID of the camera to use

`--labels <string list>` .. labels under which the objects shall be addressed e.g. "GuteLaune? jasmin". Note that of course all the following lists must be of same length.

`--plyfiles <string list>` .. files specifying geometry in PLY (Polygon File Format), e.g. "instantiations/ply-models/GuteLaune.ply instantiations/ply-models/jasmin.ply"

`--siftfiles <string list>` .. files specifying the SIFT features on the object surface. These are created with the recognizer in learn mode. e.g. "instantiations/sift-models/GuteLaune.sift instantiations/sift-models/jasmin.sift"

`--display` .. activate/deactive OpenCV display, note that this does not get updated, once the window was hidden by another window

Object Recognizer 3D Driver

The driver component is used to learn new models or to test recognition.

There is an example cast file in `subarchitectures/vision.sa/config/test-recognizer3D/single-recognizer3D.cast`

CAST command line parameters

`--labels <string list>` .. list of objects to learn/recognize (note: you can only learn one object at once), e.g. "GuteLaune? jasmin"

`--mode <mode>` .. mode can be either LEARN or RECOGNIZE (the default)

`--Loops <int>` .. don't know what that parameter means ...

Pan-Tilt-Zoom Server

The component `PanTiltZoomServer` can be used to communicate with the PTZ unit via the CAST working memory.

```
JAVA MG ptzServer ptz.ptzServer.PanTiltZoomServer
```

Command line parameter:

`--testGUI` ... starts a simple GUI to test the commands.

available enumerations / types

PTZPose

The `PTZPose` defines a pose of the PTZ Unit.

Fields:

- `pan` ... the cam pan value [rads]
- `tilt` ... the cam tilt value [rads]
- `zoom` ... the cam zoom value (does not matter for the used hardware)

PTZCompletion

- `FAILED` ... the command failed
- `SUCCEEDED` ... the command was successful
- `COMPINIT` ... initial value

available commands

If one of the following commands is added to the working memory, the PTZ-Unit will execute the corresponding action.

SetPTZPoseCommand

This command sets a pose of the PTZ unit.

Fields:

- `PTZPose pose` ... this field has to be set by the user to the target pose and it is updated after the completion of the movement with the reached pose
- `PTZCompletion comp` ... shows the completion of the command and will change by the system to `SUCCEEDED` if no errors occur

GetPTZPoseCommand

This command gets the pose of the PTZ unit.

Fields:

- `PTZPose pose` ... represents the current pose (set by the system)
- `PTZCompletion comp` ... represents the completion of the command and will change by the system to `SUCCEEDED` if no errors occur

Gazebe world files

The part of gazebo world files relevant for you is about adding and positioning models, typically located at the end of the files we provide. Let's take `simple.world` as an example.

You can mostly forget about the parts regarding physics and rendering settings. Except maybe lighting, which you might want to adjust to modify how challenging the world is for simulated object recognition.

Lighting

This provides nice and realistic shadows (on most machines anyway, there seem to be issues with some graphics cards)

```
<rendering:ogre>
  <ambient>0.8 0.8 0.8 1.0</ambient>
  <sky>
    <material>Gazebo/Grey</material>
  </sky>
</rendering:ogre>

<!-- White Directional light -->
<model:renderable name="directional_white">
  <light>
    <type>directional</type>
    <direction>0.1 -0.6 -0.4</direction>
    <diffuseColor>0.9 0.9 0.9</diffuseColor>
    <specularColor>0.7 0.7 0.7</specularColor>
                      <!-- Constant(0-1) Linear(0-1) Quadratic -->
    <attenuation>1.0 0.0 0.0</attenuation>
    <range>.8</range>
  </light>
</model:renderable>
```

This provides dull, but easier ambient only lighting

```
<rendering:ogre>
  <ambient>1.0 1.0 1.0 1.0</ambient>
  <sky>
    <material>Gazebo/Grey</material>
  </sky>
</rendering:ogre>
```

Object coordinate frames

Coordinate frames for objects, notably boxes, are defined in the following way:

- origin is in the object center
- axes are aligned with box sides. For cereal boxes this means: z points up, x points along the longer base side, y points along the shorter base side.

So a box of height h resting on the ground will have a z coordinate of h/2. An object of height h resting on a table of height H will have a z coordinate of H + h/2.

NOTE: Only the robot has its origin at z = 0, between the wheels, with x pointing forward and y pointing left.

Positioning objects in the gazebo world

To set the pose of an object, change `xyz` and `rpy` a the start of the model, e.g.

```
...
  <!-- robot model -->
  <model:physical name="robot">
    <xyz>0.0 0.0 0.01</xyz>
    <rpy>0.0 0.0 0.0</rpy>
...
  <model:physical name="cereals1_model">
    <static>false</static>
    <xyz>1 0 0.447</xyz>
    <rpy>0 0 90</rpy>
...
```

NOTE: Due to the way gazebo calculates contact forces and positions etc., you should position your objects a bit (say 0.01 m) in the air. Otherwise, if you put an object precisely on the ground, the objects might forcefully jump into the air at gazebo startup, as the physics engine tries to resolve a contact violation.

NOTE: Due to the way gazebo calculates contact forces and positions etc., objects will slightly penetrate each other (depending on how "stiff" the simulation is configured). So don't be surprised if objects are reported (by `GazeboVision` as slightly lower then they should be.

== Static and non-static objects ===

Objects with `static` set to true, will not obey gravity, but just float in mid air, but still take part in collision checks etc., e.g.

```
  <!-- some walls -->
  <model:physical name="walls_model">
    <static>true</static>
    <body:box name="walls_body">
      <geom:box name="wall_front_geom">
        <xyz>5 0 1.4</xyz>
        <size>0.1 10 2.8</size>
...
```

Adding models

Each model needs its own model name, e.g.

```
  <model:physical name="robot">
...
  <model:physical name="walls_model">
..
  <model:physical name="table1_model">
...
  <model:physical name="cereals1_model">
...
```

The actual model is typically provided via an `include` statement

```
<model:physical name="table1_model">
  <static>false</static>
  <xyz>1 0 0.176</xyz>
  <include embedded="true">
     <xi:include href="models/table.model" />
  </include>
</model:physical>
```

Adding several instances of models

UPDATE: The nasty bug described below was fixed. The patch for 0.9.0 is in the GAR installer.

NOTE: Due to a nasty bug (one of several ...) of gazebo it is **not** possible, to add the same instance of a model file several times (as the abouve include statement was intended to be used). You will have to copy and rename model files, as in

```
<model:physical name="table1_model">
  <static>false</static>
  <xyz>1 0 0.176</xyz>
  <include embedded="true">
     <xi:include href="models/table1.model" />
  </include>
</model:physical>

<model:physical name="table2_model">
  <static>false</static>
  <xyz>1 0 0.176</xyz>
  <include embedded="true">
     <xi:include href="models/table2.model" />
  </include>
</model:physical>
```

This is required essentially only for the table objects, as cereal boxes are unique anyway.