Abstract

TODO: ...
Contents

Abstract . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
Table of Contents . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
List of Figures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

1 Introduction 8
  1.1 Software Overview . . . . . . . . . . . . . . . . . . . . . . . . . 8
  1.2 Installing CASRobot . . . . . . . . . . . . . . . . . . . . . . . 8
    1.2.1 Operating System . . . . . . . . . . . . . . . . . . . . . . 8
    1.2.2 Prerequisites . . . . . . . . . . . . . . . . . . . . . . . . . 9
    1.2.3 Download CASRobot . . . . . . . . . . . . . . . . . . . . 10
    1.2.4 Install CASRobot . . . . . . . . . . . . . . . . . . . . . . 11

2 CASRobot 12
  2.1 Data <casrobot/data.h> . . . . . . . . . . . . . . . . . . . . . . . 12
    2.1.1 Time . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
    2.1.2 Point2D . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
    2.1.3 Point3D . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
    2.1.4 Size2D . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
    2.1.5 Size3D . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
    2.1.6 ColouredPoint . . . . . . . . . . . . . . . . . . . . . . . . 12
    2.1.7 Colour4f . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
    2.1.8 Orientation3D . . . . . . . . . . . . . . . . . . . . . . . . . 13
    2.1.9 Pose2D . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
    2.1.10 Pose3D . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
    2.1.11 PanTiltZoom . . . . . . . . . . . . . . . . . . . . . . . . . 13
    2.1.12 Data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
    2.1.13 RobotImage . . . . . . . . . . . . . . . . . . . . . . . . . . 13
    2.1.14 LaserScan . . . . . . . . . . . . . . . . . . . . . . . . . . 13
    2.1.15 PointCloud . . . . . . . . . . . . . . . . . . . . . . . . . . 13
    2.1.16 Snap . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
    2.1.17 RobotHistory . . . . . . . . . . . . . . . . . . . . . . . . . 14
  2.2 Memory Management . . . . . . . . . . . . . . . . . . . . . . . . 14
    <casrobot/objectmanagement.h> . . . . . . . . . . . . . . . . . . 14
      2.2.1 ManagedObject . . . . . . . . . . . . . . . . . . . . . . . 14
      2.2.2 Handle . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
  2.3 Logging <casrobot/logger.h> . . . . . . . . . . . . . . . . . . . . 14
    2.3.1 Logger . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
    2.3.2 Delogger . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
  2.4 Threads <casrobot/thread.h> . . . . . . . . . . . . . . . . . . . . 15
3 Robot 17
   3.1 Components .................................. 17
   3.2 Libraries ...................................... 17
      3.2.1 Example .................................. 17
      3.2.2 Configuration .............................. 17
   3.3 Devices ....................................... 17
      3.3.1 Cameras .................................... 18
   3.4 Tasks .......................................... 19
   3.5 Foreground Tasks ............................... 19
   3.6 Background Tasks ............................... 19
   3.7 Autonomy Modules ............................... 19
      3.7.1 Configuration .............................. 19
      3.7.2 Arguments ................................ 20
   3.8 Example ....................................... 20
   3.9 Configuration ................................. 20
      3.9.1 Attributes ................................ 20
      3.9.2 Children ................................. 21
   3.10 Util ......................................... 21
      3.10.1 Example ................................ 21
      3.10.2 Configuration ............................ 21

4 RobotClient 24
   4.1 Example ....................................... 24
   4.2 Configuration ................................. 24
      4.2.1 Attributes ................................ 24
      4.2.2 Children ................................. 24
   4.3 ICE ........................................... 24

5 RobotServ 25
   5.1 Example ....................................... 25
   5.2 Configuration ................................. 25

6 RobotGUI 26
   6.1 Example ....................................... 26
   6.2 Configuration ................................. 26
   6.3 Maps .......................................... 27
   6.4 GUI ............................................ 27
   6.5 Libraries ...................................... 27
      6.5.1 Example ................................ 27
      6.5.2 Configuration ............................ 27
   6.6 Panels ........................................ 27
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.6.1</td>
<td>Image Panel</td>
<td>27</td>
</tr>
<tr>
<td>6.6.2</td>
<td>Scroll Panel</td>
<td>28</td>
</tr>
<tr>
<td>6.6.3</td>
<td>Split Panel</td>
<td>29</td>
</tr>
<tr>
<td>6.6.4</td>
<td>Tab Panel</td>
<td>29</td>
</tr>
<tr>
<td>6.6.5</td>
<td>Map Panel</td>
<td>30</td>
</tr>
<tr>
<td>6.6.6</td>
<td>Snap View Panel</td>
<td>31</td>
</tr>
<tr>
<td>6.6.7</td>
<td>Camera Panel</td>
<td>31</td>
</tr>
<tr>
<td>6.6.8</td>
<td>Robot Panel</td>
<td>33</td>
</tr>
<tr>
<td>6.7</td>
<td>Renders</td>
<td>36</td>
</tr>
<tr>
<td>6.7.1</td>
<td>Crosshair Render</td>
<td>36</td>
</tr>
<tr>
<td>6.7.2</td>
<td>Camera Render</td>
<td>37</td>
</tr>
<tr>
<td>6.7.3</td>
<td>Speed Render</td>
<td>37</td>
</tr>
<tr>
<td>6.7.4</td>
<td>Power Render</td>
<td>38</td>
</tr>
<tr>
<td>6.7.5</td>
<td>Tilt Render</td>
<td>39</td>
</tr>
<tr>
<td>6.7.6</td>
<td>Overlay Render</td>
<td>39</td>
</tr>
<tr>
<td>6.7.7</td>
<td>Preset Render</td>
<td>41</td>
</tr>
<tr>
<td>6.7.8</td>
<td>Target Render</td>
<td>41</td>
</tr>
<tr>
<td>7</td>
<td>Devices</td>
<td>43</td>
</tr>
<tr>
<td>7.1</td>
<td>Player</td>
<td>43</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Example</td>
<td>43</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Configuration</td>
<td>43</td>
</tr>
<tr>
<td>7.1.3</td>
<td>Camera</td>
<td>44</td>
</tr>
<tr>
<td>7.1.4</td>
<td>Laser</td>
<td>44</td>
</tr>
<tr>
<td>7.1.5</td>
<td>Pos1d</td>
<td>44</td>
</tr>
<tr>
<td>7.1.6</td>
<td>Pos2d</td>
<td>44</td>
</tr>
<tr>
<td>7.1.7</td>
<td>Pos3d</td>
<td>44</td>
</tr>
<tr>
<td>7.1.8</td>
<td>Power</td>
<td>45</td>
</tr>
<tr>
<td>7.1.9</td>
<td>PTZ</td>
<td>45</td>
</tr>
<tr>
<td>7.2</td>
<td>Delogger</td>
<td>45</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Example</td>
<td>45</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Configuration</td>
<td>45</td>
</tr>
<tr>
<td>7.3</td>
<td>Cameras</td>
<td>46</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Fake</td>
<td>46</td>
</tr>
<tr>
<td>7.3.2</td>
<td>CASCam</td>
<td>46</td>
</tr>
<tr>
<td>7.3.3</td>
<td>NightSwiss</td>
<td>47</td>
</tr>
<tr>
<td>7.4</td>
<td>Lasers</td>
<td>48</td>
</tr>
<tr>
<td>7.4.1</td>
<td>UTM Laser</td>
<td>48</td>
</tr>
<tr>
<td>8</td>
<td>Tasks</td>
<td>50</td>
</tr>
<tr>
<td>8.1</td>
<td>Landmark/Victim Location</td>
<td>50</td>
</tr>
<tr>
<td>8.1.1</td>
<td>CollectSnap</td>
<td>50</td>
</tr>
<tr>
<td>8.2</td>
<td>Position Tracking</td>
<td>50</td>
</tr>
<tr>
<td>8.2.1</td>
<td>MBPosTrack</td>
<td>51</td>
</tr>
<tr>
<td>8.2.2</td>
<td>ICPPosTrack</td>
<td>53</td>
</tr>
<tr>
<td>8.3</td>
<td>Autonomy</td>
<td>54</td>
</tr>
<tr>
<td>8.3.1</td>
<td>LocalMap</td>
<td>54</td>
</tr>
<tr>
<td>8.3.2</td>
<td>BasicController</td>
<td>55</td>
</tr>
<tr>
<td>8.3.3</td>
<td>LocalPlanner</td>
<td>56</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>An overview of the CASRobot programs and their communication channels.</td>
<td>8</td>
</tr>
<tr>
<td>6.1</td>
<td>An image panel.</td>
<td>28</td>
</tr>
<tr>
<td>6.2</td>
<td>An image panel within a scroll panel.</td>
<td>28</td>
</tr>
<tr>
<td>6.3</td>
<td>A split panel with a pair of images.</td>
<td>29</td>
</tr>
<tr>
<td>6.4</td>
<td>A tab panel with two tabs.</td>
<td>30</td>
</tr>
<tr>
<td>6.5</td>
<td>A map of an autonomous victim search.</td>
<td>31</td>
</tr>
<tr>
<td>6.6</td>
<td>A snap panel displaying an autonomously found victim.</td>
<td>32</td>
</tr>
<tr>
<td>6.7</td>
<td>A camera panel showing a night-vision view of a SwissRanger.</td>
<td>32</td>
</tr>
<tr>
<td>6.8</td>
<td>A robot panel showing a wide angle camera with a number of renderers.</td>
<td>33</td>
</tr>
<tr>
<td>6.9</td>
<td>Crosshair rendered on a robot panel.</td>
<td>36</td>
</tr>
<tr>
<td>6.10</td>
<td>Speed render showing speed for robot motions.</td>
<td>37</td>
</tr>
<tr>
<td>6.11</td>
<td>Power render showing robot’s battery voltage.</td>
<td>38</td>
</tr>
<tr>
<td>6.12</td>
<td>Tilt render showing the roll and pitch of the robot.</td>
<td>39</td>
</tr>
<tr>
<td>6.13</td>
<td>Overlay of thermal image.</td>
<td>40</td>
</tr>
<tr>
<td>6.14</td>
<td>Current configuration of the robot and some available preset positions.</td>
<td>41</td>
</tr>
<tr>
<td>6.15</td>
<td>Target render of SwissRanger field of view.</td>
<td>42</td>
</tr>
<tr>
<td>7.1</td>
<td>Side by side images from a SwissRanger and a NightSwiss camera.</td>
<td>47</td>
</tr>
<tr>
<td>7.2</td>
<td>A Hokuyo UTM-30LX scanning laser range finder.</td>
<td>48</td>
</tr>
<tr>
<td>8.1</td>
<td>Local map used by MBPosTrack for localisation.</td>
<td>51</td>
</tr>
<tr>
<td>9.1</td>
<td>GeoTIFF image of an autonomous run made by Emu during the Thailand Open 2009</td>
<td>59</td>
</tr>
<tr>
<td>9.2</td>
<td>FastSLAM map of the level 3 maze.</td>
<td>60</td>
</tr>
<tr>
<td>11.1</td>
<td>Negotiator, who came 2nd in the Mobility class in the 2009 RoboCup Rescue</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>Robot League competition.</td>
<td></td>
</tr>
<tr>
<td>11.2</td>
<td>Emu, who came 1st in the Autonomy class in the 2009 RoboCup Rescue Robot</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>League competition.</td>
<td></td>
</tr>
<tr>
<td>11.3</td>
<td>Flinders and Wills resting after a big day.</td>
<td>65</td>
</tr>
<tr>
<td>11.4</td>
<td>The Pioneer AT preparing to conquer new terrain.</td>
<td>66</td>
</tr>
<tr>
<td>11.5</td>
<td>PeopleBot awaiting instructions.</td>
<td>66</td>
</tr>
<tr>
<td>11.6</td>
<td>CASTER, the 2005 Team CASualty robot, which came 3rd in the 2005 RoboCup</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>Rescue Robot League competition.</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

TODO: …

1.1 Software Overview

TODO: …

Figure 1.1: An overview of the CASRobot programs and their communication channels.

1.2 Installing CASRobot

1.2.1 Operating System

CASRobot has been specifically developed to run under Linux. It is known to run on Ubuntu 9.10 and has been made to work on Debian Lenny.
1.2.2 Prerequisites

CASRobot requires a number of libraries in order to compile and run. The more important ones are described later in this section.

The following command can be used in Ubuntu 9.10 to bring in many of the required packages as well as a number of helpful utilities.

```bash
sudo aptitude install build-essential openssh-server cvs subversion netpbm libdc1394-22-dev coriander libzeroc-ice33-dev libtool libpango1.0-dev lib64asound2-dev cmake libgststreamer0.10-dev libgtk2.0-dev libxine1 libxine-dev liblzo2-dev liblapack-dev libtiff-tools geotiff-bin libgeotiff-dev libxv-dev libncurses5-dev libavutil-dev libavutil-extra-49 libavcodec-dev libavcodec-extra-52 libbscale-dev libbscale-extra-0 libavformat-dev libavformat-extra-52 libpostproc-dev libpostproc-extra-51 libavdevice-dev libavdevice-extra-52 libavfilter-dev libavfilter-extra-0 ffmpeg qt4-dev-tools libboost-dev freeglut3-dev texlive
```

More recent versions of some prerequisites can be found at:


Qt

Qt 4 is used by RobotGUI to provide the various panels and widgets used to construct the user interface. It also provides access to OpenGL which is used for drawing the map and some other 3D rendering. If you are only wanting to build RobotServ then Qt is not required.

Ice

CASRobot uses Ice 3.3 for communication between RobotServ and robot clients.

**NOTE:** Debian currently only provides Ice 3.2 in its repositories so you will need to build it from source yourself.

OpenCV

OpenCV is used to provide some GUI options for RobotServ. The version in the standard repositories is not updated regularly but is acceptable for the basic GUI stuff it's used for in RobotServ. If you want to do reasonably advanced image processing you will likely need to use a more recent version which will have to be built from source.

Player

CASRobot uses Player to provide low level drivers for accessing a variety of hardware devices. The Ubuntu and Debian repositories currently offer Player 2.0.4. This version does work with CASRobot but we recommend you upgrade to Player 2.0.5 where possible.

CASRobot cannot as yet work with Player 2.5 or later.
**NewMat**

CASRobot uses the NewMat library for matrix calculations. To install NewMat:

2. Extract source files
3. Change to source directory
4. Call `sudo make install` to install into /usr/local

**FFmpeg**

FFmpeg is a collection of libraries and utilities for the recording, converting and streaming of video and audio. LibCAS uses FFmpeg to perform video compression of camera images before sending them over the network.

**NOTE:** On Ubuntu 9.10 we need the FFmpeg extra packages (eg libavutil-extra-49, libavformat-extra-52 and libavcodec-extra-52) as the standard packages don’t include the video encoders that LibCAS uses.

**NOTE:** On Debian the FFmpeg packages in the repositories are missing the video encoders required by LibCAS so you will need to build it from source.

**LibCAS**

LibCAS is a small library used by CASRobot for image format conversions, image capture and image streaming.

To install LibCAS:

1. Check it out of subversion repository:

   `svn co http://robolab.cse.unsw.edu.au/svn/rescue/trunk/libcas`

2. Change to source directory
3. Call `sudo make install` to install into /usr/local

**NOTE:** LibCAS can capture images from a variety of cameras. By default many of these are disabled. To enable a camera you need to make changes to libcas/Makefile to activate disabled cameras. These activated cameras may need specialised libraries to be installed on the system.

**1.2.3 Download CASRobot**

CASRobot can be downloaded using the command:

`svn co http://robolab.cse.unsw.edu.au/svn/rescue/trunk/casrobot`
1.2.4 Install CASRobot

Once you have CASRobot downloaded and all prerequisites installed you can install CASRobot by:

1. Change to source directory

2. Call `sudo make install` to install into `/usr/local`

This will install the CASRobot libraries, RobotServ and RobotGUI, as well as the headers for developing with CASRobot into `/usr/local`
Chapter 2

CASRobot

The CASRobot library has been designed to be reasonably lightweight and to reduce the number of libraries it depends on. It has defined a number of classes and functions to store and process data and provide useful functionality. Some of these are described in this chapter.

2.1 Data <casrobot/data.h>

The header file casrobot/data.h defines all of the standard data types used throughout the CASRobot library.

2.1.1 Time

The Time struct defines a time given in seconds and milliseconds from a specific time. (Usually 1 January 1970)

2.1.2 Point2D

A Point2D object represents a specific point on a 2-dimensional plane.

2.1.3 Point3D

A Point3D object represents a specific point in a 3-dimensional space.

2.1.4 Size2D

A Size2D object indicates the size of a 2-dimensional object.

2.1.5 Size3D

A Size3D object indicates the size of a 3-dimensional object.

2.1.6 ColouredPoint

A point in 3 dimensions that has a specific colour.
2.1.7 Colour4f
A colour specified by red, green, blue and alpha values.

2.1.8 Orientation3D
The orientation of an object in 3 dimensions, given by roll, pitch and yaw values.

2.1.9 Pose2D
A Pose2D object represents the location and orientation of an object in 2 dimensions.

2.1.10 Pose3D
A Pose3D object represents the location and orientation of an object in 3 dimensions.

2.1.11 PanTiltZoom
A PanTiltZoom object represents the pan, tilt and zoom setting on a PTZ device attached to a camera.

2.1.12 Data
The Data class is a virtual superclass of the larger data structures that are commonly used within the CASRobot library. It provides the larger classes with automated memory management and timestamp functions.

2.1.13 RobotImage
A RobotImage can be an image frame from a camera or generated image. It is a wrapper around a casimage_frame_t object from the LibCAS library (see section 1.2.2) which is used to provide a number of image formats and conversion functions.

2.1.14 LaserScan
A LaserScan is a series of laser ranges recorded as a laser sweeps through a plane.

2.1.15 PointCloud
A PointCloud is a set of 3D points (they may or may not be coloured) usually representing points belonging to obstacles sensed by either a laser or a range camera.

2.1.16 Snap
A Snap object represents the location of an object to be recorded in a map. Snaps usually represent things such as victims or landmarks (e.g., stairs, barrels, etc).
2.1.17 RobotHistory

A RobotHistory is a series of points which the robot has already visited.

2.2 Memory Management  
<casrobot/objectmanagement.h>

When using large objects such as images and point clouds good memory management is essential because memory leaks will quickly exhaust a system’s resources. To assist in this some helpful classes are available. These classes are used to keep track of the number of references to an object and when all references are removed to automatically delete it.

2.2.1 ManagedObject

A ManagedObject keeps track of how many handles are currently pointing to it. When no more handles are pointing to a ManagedObject it automatically calls delete on itself. Except for on extremely rare occasions you will not interact directly with a ManagedObject but with a Handle to the object.

2.2.2 Handle

A Handle is essentially a pointer to a ManagedObject.

2.3 Logging  <casrobot/logger.h>

Because robots are notoriously fragile and expensive and so only accessible in small numbers it is useful when developing and testing components to be able to write to and read from logs. For this reason a number of logging classes are available.

2.3.1 Logger

A logger is used to write to a specific log. Messages written to a log can be are displayed on stdout depending on the messages’ importance. It is possible to write things such as RobotImage and PointCloud objects to a log. Usually you will be writing to casrobot::logger.

2.3.2 Delogger

Logging data is useful for understanding what happens when something breaks. Replaying a recorded robot session is accomplished using a Delogger. This allows all devices recorded during a session to be replayed so that components can be tested as though they were running on a real robot.

NOTE: Real devices have different memory requirements and CPU demands than devices reading from logs.
2.4 Threads <casrobot/thread.h>

Because a robot has to be process several things concurrently it needs to be able run several threads. For this reason we provide classes that be be used to create new threads as needed. To prevent data corruption caused by multiple threads reading and writing to the same piece of data some other classes are available to allow synchronisation between threads.

2.4.1 Thread

The Thread class can be extended to create a new thread which can then be started when required.

2.4.2 Mutex

A Mutex object can be used to protect a section of data so that only one thread can access it at a time. Then the thread is finished with the data it can unlock so another thread can use it.

2.4.3 Lock

A Lock object is used to lock a mutex and to automatically unlock it when a function returns. The Lock object unlocks the mutex when it is destroyed, which if it is a local variable inside a function happens when the function returns or an uncaught exception is thrown.

2.4.4 Semaphore

Semaphores are used to synchronise multiple threads.

2.5 Matrices <casrobot/matrices.h>

Tasks require points to be located in specific frames of reference. Converting between reference frames is done using matrices. So some matrices classes are provided. Currently CASRobot uses NewMat for its matrices.

2.5.1 ColumnVector

A ColumnVector is a single column in a matrix. They are commonly used to represent a 3D point while converting between frames of reference.

2.5.2 Matrix

A Matrix is essentially an array of arrays of numbers. In CASRobot we usually use a 4x4 matrix or sequence of them to transform a point from one frame of reference to another.
2.6 OpenCV <casrobot/opencv.h>

Sometimes is useful to display an image from within robotserv when the GUI is unavailable. For this reason several tasks use OpenCV to open windows on the robot’s screen. Unfortunately OpenCV has problems when windows are opened in multiple threads. So we have added an OpenCV class that can be used to open an OpenCV window. This class will then open required windows in its own thread.
Chapter 3

Robot

Because CASRobot is a library for operating a robot the Robot class is included. This can be used to access the various components that make up the robot.

3.1 Components

For the most part the components of a robot are separated into two subclasses, devices and tasks. Devices for the most part are the various pieces of hardware that make up the robot. Tasks are processes that are to be performed using the devices.

3.2 Libraries

If you wish to use a new component on a robot it is possible to write a new driver for a device or to implement a new task in a library. A configuration file can then load that library and use the new robot components. Chapter 10 goes into more detail of how to develop a library for CASRobot.

NOTE: Configuration files are mostly parsed in the order of options within the file. So libraries required to be loaded should be at the top of a configuration file.

3.2.1 Example

<library file="libexample.so" />

3.2.2 Configuration

Attributes

file - the shared object file that holds the library

3.3 Devices

A device can be any piece of hardware attached to a robot. Things such cameras, laser and servo motors are all devices that CASRobot can be used to listen to
and control.

CASRobot is able to work with a number of devices. Some of the devices that are built into the standard CARobot library can be found in chapter 7.

### 3.3.1 Cameras

As part of the original reason for developing CASRobot was to operate robots in situations that are too dangerous for humans cameras are often used to allow a remote operator to perceive the environment around the robot. So they are an important part of CASRobot.

#### Example

```xml
<camera name="Camera" type="CASCam" subtype="UVC">
    <pose />
    <fov />
</camera>
```

**Streaming Camera**

Streaming cameras are a subtype of camera specifically developed to be able to stream their images over a network to a client, such as a GUI in front of an operator.

#### Example

```xml
<camera name="Camera" type="CASCam" subtype="UVC">
    <stream />
</camera>
```

**Configuration Attributes**

- **log** - whether or not to log the camera data, overridden by child log tag (default false)
- **Children**
  - **vault** - sets the size of the memory for the camera
    - **history** - how much image history that should be cached (default 1)
  - **pose** - sets the camera’s pose, this allows relationships to be established between pixes on separate cameras if their poses are from a single point
    - **x** - the x position of the camera (default 0)
    - **y** - the x position of the camera (default 0)
    - **z** - the x position of the camera (default 0)
    - **roll** - the roll of the camera (default 0)
    - **pitch** - the pitch of the camera (default 0)
    - **yaw** - the yaw of the camera (default 0)
  - **fov** - the field of view for the camera
x - the angular width of the camera in degrees (default 0)
y - the angular height of the camera in degrees (default 0)

log -
log - whether or not to log the camera data (default true)
frequency - says to log only every Nth image, 0 logs all images (default 0)

stream - sets parameter controlling the image stream to a client
stream - allows the streaming to be turned off (default true)
delay - sets how many milliseconds to wait between placing images in the stream (default 200)

3.4 Tasks

Tasks are used to process sensor data from a robot’s devices in order to accomplish its goals. Tasks can be used to provide a robot with visual object identification, autonomous navigation and other abilities.

CASRobot separates tasks into 2 sub groups foreground and background tasks.

Some basic tasks are provided in the standard CASRobot library and are described in chapter 8.

3.5 Foreground Tasks

A robot can only have 1 foreground task active at any point in time. If several foreground tasks need to be executed they are queued and run one at a time.

3.6 Background Tasks

A robot can have as many background tasks as it needs running concurrently. Each background task runs in its own thread.

3.7 Autonomy Modules

Autonomy modules are background tasks with some extra functionality. Autonomy tasks can be automatically paused while a foreground task is running.

3.7.1 Configuration

speed - sets the maximum forward speed for the robot
turnrate - sets the maximum angular speed for the robot
delay - sets how long to wait before deciding the network connection to a client has broken and the robot should autonomously navigate to a location with better reception
3.7.2 Arguments

Autonomy modules can accept the following arguments to the `processArgs` function:

- **pause** - stop autonomous motion
- **resume** - resume/start autonomous motion
- **toggle** - stop/start autonomous motion
- **scan on** - enable autonomous victim/landmark scans
- **scan off** - disable autonomous victim/landmark scans
- **left wall** - follow the left wall
- **right wall** - follow the rights wall
- **straight** - go straight until you encounter an obstacle
- **bump left** - go straight until you encounter an obstacle, then turn left until you can move again
- **bump right** - go straight until you encounter an obstacle, then turn right until you can move again
- **recover radio** - autonomously navigate if connection breaks
- **ignore radio** - do nothing if connection breaks
- **ping** - network connection is fine

3.8 Example

```xml
<robot name="Pioneer">
  <player host="127.0.0.1">
    <pos2d name="Drive" id="0" />
    <laser name="Laser" id="0" />
  </player>

  <background name="MBICPosTrack" />
</robot>
```

3.9 Configuration

3.9.1 Attributes

- **name** - the name of the robot. Each robot should have its own name.
- **type** - the type of robot (default CASRobot)
3.9.2 Children

The CASRobot type of robot can then accept these options:

- **library** - loads a library for CASRobot extensions, see section 3.2
- **player** - opens a connection to Player, see section 7.1
- **device** - creates a driver for a device, see chapter 7 for available devices
- **camera** - creates a camera driver, see section 7.3 for available cameras
- **task** - creates a task, could be either foreground or background, see chapter 8 for available tasks
- **background** - creates a background task, see chapter 8 for available background tasks
- **foreground** - creates a foreground task, see chapter 8 for available foreground tasks
- **delog** - creates a delogger to replay logged devices, see section 7.2 for options
- **client** - creates a robot client to connect to RobotServ, see chapter 4 for options
- **util** - configures the utility class for this robot, see section 3.10 for options

3.10 Util

Every robot has its own util object that provides methods for getting information about the robot. These functions can be used to get the robot's state, grab a point cloud for processing and to get the kinematics for various sensors.

3.10.1 Example

```xml
<util laserPose="kinLaser">
  <laser id="Laser" min="0.5" max="3.9" />
  <kinematics name="kinLaser">
    <tx x="0.20" />
    <ty y="0.15" />
  </kinematics>
</util>
```

3.10.2 Configuration

Attributes

- **mainCam** - the name of the camera to provide colours for point clouds
- **rangeCam** - the name of the camera to provide depth images for point clouds
- **odometry** - the name of a Pos3D or Pos2D device to provide odometry
- **robotAtt** - the name of a Pos3D device to provide the attitude of the robot
headAtt - the name of a Pos3D device to provide the attitude of the head

debug - whether or not to print extra debugging information (default false)

cache - the number of milliseconds before a cached robot state is too old (default 500)

optimise - whether or not to print extra debugging information (default true)

sensorPose - the name of a kinematics chain to provide the sensor pose

laserPose - the name of a kinematics chain to provide the laser pose

Children

laser - defines the laser to be used for mapping and its location on robot

  name - the name of the laser device to use for mapping/localisation
  id - same as name
  min - laser readings smaller than this should be ignored (default 0.5)
  max - laser readings larger than this should be ignored (default 3.8)

kinematic - A kinematics chain

  Attributes

  name - a name for the kinematics chain

Children The attributes of the children of a kinematics tag can be written as a mathematical operation. See section 3.10.2 for details.

dh - DH kinematic element

  alphai - (default 0)
  ai - (default 0)
  di - (default 0)
  thetai - (default 0)

tax - translation along X axis
  x - the amount of translation along X axis (default 0)

ty - translation along Y axis
  y - the amount of translation along Y axis (default 0)

tz - translation along Z axis
  z - the amount of translation along Z axis (default 0)

rx - rotation about X axis in degrees
  x - the amount of rotation about X axis (default 0)
  pitch - the same as x

ry - rotation about Y axis in degrees
  y - the amount of rotation about Y axis (default 0)
  roll - the same as y

rz - rotation about Z axis in degrees
z - the amount of rotation about Z axis (default 0)

yaw - the same as z

offset - a 3D translation without rotation

x - the amount of translation along X axis (default 0)
y - the amount of translation along Y axis (default 0)
z - the amount of translation along Z axis (default 0)

rotate - a single rotation about an axis, only 1 attribute allowed

roll - the amount of translation along Y axis (default 0)
pitch - the amount of translation along X axis (default 0)
yaw - the amount of translation along Z axis (default 0)

chain - a chain that is part of a larger chain

name - the name of a previously defined chain

kinematics - same as kinematic

Kinematics Values

The values for a kinematics element can be a number, a constant defined in the util tag, PI or a value retrieved from a Pos1D, Pos2D, Pos3D or PTZ device.

Example

- "1"
- "1 + 5"
- "4 * ( 1 + 2 )"
- "PI"
- "-PI/2"
- "LaserXDisplacement"
- "PTU:pan"
- "Arm:x"
- "RobotAtt:roll"
- "PTU:tilt - Arm:x"
Chapter 4

RobotClient

A robot client is a part of a robot that provides access to the components of another robot as though they were components of this robot.

4.1 Example

<client host="Emu" />

4.2 Configuration

4.2.1 Attributes

log - whether on not to log data received by the client (default false)
endport - the port Ice should use for the client (default -1)

4.2.2 Children

log - whether on not to log data received by the client (default false)

log - whether on not to log data received by the client (default true)
frequency - only save every Nth image (default 0)

4.3 ICE

CASRobot uses [ICE] to communicate between robot clients and RobotServ.
Chapter 5

RobotServ

RobotServ is a utility program that wraps a robot and listens on a specified port for commands from robot clients.

5.1 Example

```xml
<robot name="Pioneer" port="10101" >
  <player host="127.0.0.1" >
    <pos2d name="Drive" id="0" />
    <laser name="Laser" id="0" />
  </player>

  <background name="MBPosTrack" />
</robot>
```

5.2 Configuration

As RobotServ wraps a robot it takes all configuration options for a robot (see section 3.9) and add one more optional attribute:

- **port** - the port to listen for clients on (default 10101)
Chapter 6

RobotGUI

RobotGUI is a program that provides user interfaces for robots using the CAS-Robot library. The user interface can be used to observe and to control a robot. It also provides the ability to map an environment as a robot explores it and to display the map to the user.

6.1 Example

```xml
<robotgui>
  <library file="libexample.so" />

  <robot name="Robot">
    <client host="robot" />
  </robot>

  <map name="Map" type="FastSLAM" robot="Robot" />

  <gui>
    <mapview map="Map" />
  </gui>
</robotgui>
```

6.2 Configuration

**Children**

- **library** - loads a dynamic library to extend CASRobot

- **robot** - creates/connects to a robot to be controlled by the GUI, see chapter 3 for configuration options for robots

- **map** - creates a map, see chapter 9 for available maps

- **gui** - defines the GUI elements to be used
6.3 Maps

RobotGUI is able to instantiate a number of maps that can be used to record the environment a robot moves through. See chapter 9 for available maps.

6.4 GUI

The actual user interface is a window with a single panel. This panel may or may not include a number of subpanels depending on the configuration options. Section 6.6 describes a large number of panels that are available with CASRobot.

6.5 Libraries

Just like it is possible to add new devices and tasks via a library (see section 3.2), it is also possible to add new panels and renders with a library. See chapter 10 for instructions on how to build a library.

**NOTE:** Configuration files are mostly parsed in the order of options within the file. So libraries required to be loaded should be at the top of a configuration file.

6.5.1 Example

```xml
<library file="libexample.so"/>
```

6.5.2 Configuration

Attributes

- `file` - the shared object file that holds the library

6.6 Panels

The actual GUI for RobotGUI is comprised of any number of panels. These are some of the panels that you can use to construct your user interface.

6.6.1 Image Panel

An image panel is used to display a single image. The minimum size of the panel is the size of the image so large images can make the window larger than the screen it is on.

Example

```xml
<image file="eagle.jpg"/>
```
Configuration

file - the location of the file to display

image - same as file

src - same as file

6.6.2 Scroll Panel

A scroll panel is used to allow a large panel to fit inside an area smaller than the panel’s minimum size. The scrolls bars on the edge of the scroll panel can be used to change the visible portion of the internal panel.
Configuration
Scroll panels do not currently have attributes that can be set.

6.6.3 Split Panel
A split panel allows two or more panels to be placed side by side. Split panels can be orientated horizontally or vertically.

![Figure 6.3: A split panel with a pair of images.]

Example

```xml
<split orientation="horizontal">
  <image src="eagle-small.jpg" />
  <scroll>
    <image src="eagle.jpg" />
  </scroll>
</split>
```

Configuration
orientation - the orientation of the split panel, can be horizontal (default) or vertical

6.6.4 Tab Panel
A tab panel allows one or more panels to be layered on top of each other with the top most panel selected using tabs.

Example

```xml
<tabs>
  <tab name="Image" icon="eagle-small.jpg">
    <image src="eagle-small.jpg" />
  </tab>
</tabs>
```
Figure 6.4: A tab panel with two tabs.

```xml
<scroll>
  <image src="eagle.jpg" />
</scroll>
</tabs>

Configuration

name - the textual label to use on the tab

icon - the name of an image file to use as an icon for the tab

6.6.5 Map Panel

A map panel shows the current state of a map. The top section shows the map and if selected can be used to move around the map. The bottom section allows the map to be saved, previous maps to be loaded and more.

It is possible to move around the drawn map. The W, A, S, D, Q and E keys move forward, left, back, right, up and down. The arrow keys can be used to rotate the view. If needed the R key can be used to reset the view to its original position.

Example

```xml
<mapview map="Map" />
```

Configuration

map - the name of the map to show in the panel
6.6.6 Snap View Panel

A snap view panel is used to view the snaps in a map. Snaps are listed by their statuses and when selected their stored images are shown in the field at the bottom. The snap view panel can be used to change the status of a snap.

Example

```xml
<snapview map="Map" />
```

Configuration

- `map` - the name of the map whose snaps are to show in the panel
- `robot` - the name of a robot to pause while a snap is confirmed
- `task` - the name of the autonomy task on the robot to be paused/resumed

6.6.7 Camera Panel

A camera panel is used to display a camera on a robot with no interaction.

Example

```xml
<camera robot="Robot" camera="Camera" />
```
Figure 6.6: A snap panel displaying an autonomously found victim.

Figure 6.7: A camera panel showing a night-vision view of a SwissRanger.
Configuration

robot - the name of a robot

camera - the name of the camera on the robot to display

6.6.8 Robot Panel

A robot panel is used to display the state of a robot and allow an operator to control it.

Figure 6.8: A robot panel showing a wide angle camera with a number of renderers.

If the drive has been set then the robot can be driven using the W, A, S and D keys. When a PTZ is given and the main camera FOV is set then left-clicking on the robot panel will move the PTZ so the camera is looking at the selected point. Right-clicking sets the PTZ to (0, 0, 0). Scrolling with a mouse wheel in the robot panel increases or decreases the speed the robot will use in response to the driving keys. Holding the Shift key while scrolling with the mouse wheel will change the turn rate for the robot.

Example

<robotview robot="Emu" camera="Wide" ptz="PTZ">
  <drive pos2d="Drive" />
</robotview>

<!-- Standard -->
Configuration

Attributes

robot - the name of a robot
camera - the name of the camera on the robot to display
ptz - the name of the PTZ device to move/zoom the camera
drive - the name of a Pos2D device that can be used to drive the robot

Children

drive - configures commands for driving the robot

pos2d - the name of a Pos2D on the robot to be used to drive it
drive - the same as pos2d
name - the same as pos2d
speed - the maximum forward/reverse speed of the robot

turnrate - the maximum turnrate for the robot

step - the fraction of the maximum speed to alter the current speed when mouse wheel is moved

fgtask - configures a key stroke to call execute/interrupt on a foreground task on the robot

key - the key which when pressed performs the action

modifiers - modifier keys (eg Alt, Ctrl) which are required in conjunction with the action key

interrupt - if true sets the action to interrupt the current foreground task

task - the name of the task to execute

args - a space separated list of arguments for the execute command

arg - a single argument for the execute command

Children

arg - defines a single argument for the execute command

value - the value of the argument for the execute command

arg - the same as value

bgtask - configures a key stroke to call processArgs on a background task on the robot

key - the key which when pressed performs the action

modifiers - modifier keys (eg Alt, Ctrl) which are required in conjunction with the action key

task - the name of the task to process the arguments

args - a space separated list of arguments for processing

arg - a single argument for processing

Children

arg - defines a single argument for processing

value - the value of the argument for processing

arg - the same as value

pos1d - configures a pair of keys to control a Pos1D device on the robot

pos1d - the name of the Pos1D device to be controlled

name - the same as pos1d

speed - the speed with which to move the device

step - if 0 (default) the device is moved using velocity commands if non-zero the devices is moved by setting a desired position and step is the size of the increment to the desired position

min - the minimum position the device can be set to
max - the maximum position the device can be set to
left - the key that is pressed to increment the device position
right - the key that is pressed to decrement the device position

**Renders** - a number of renders can be added to the robot panel, available renders are described in section 6.7

### 6.7 Renders

#### 6.7.1 Crosshair Render

The crosshair render is used to draw a crosshair on the robot panel.

![Crosshair rendered on a robot panel.](image)

Figure 6.9: Crosshair rendered on a robot panel.

**Example**

```xml
<crosshair bounds="0.2 0.2 0.6 0.6" colour="1 1 0" />
```

**Configuration**

**Attributes**

invertX - whether or not to invert the X axis (default false)
invertY - whether or not to invert the Y axis (default false)
rotate - rotation in degrees to be applied to the render (default 0)
6.7.2 Camera Render

A camera render is used to toggle the display of a camera on the robot panel.

Example

```xml
<camera name="Zoom" toggle="z" />
```

Configuration

Attributes

- invertX - whether or not to invert the X axis (default false)
- invertY - whether or not to invert the Y axis (default false)
- rotate - rotation in degrees to be applied to the render (default 0)
- bounds - the bounding box for the render, given as fractions of the panel, given as left bottom width height
- camera - the name of the camera to be shown
- name - the same as camera
- toggle - a key that will toggle whether or not to display the camera render

6.7.3 Speed Render

The speed render shows the speed and turnrate that the robot will move at in response to the driving keys in the robot panel. Speeds are shown as a fraction of the maximum speed. The displayed speed is NOT the speed the robot will move at when under autonomous control.

![Speed render](image)

Figure 6.10: Speed render showing speed for robot motions.

Example

```xml
<speed bounds="0.25 0.9 0.5 0.1" />
```
Configuration

Attributes

invertX - whether or not to invert the X axis (default false)

invertY - whether or not to invert the Y axis (default false)

rotate - rotation in degrees to be applied to the render (default 0)

bounds - the bounding box for the render, given as fractions of the panel, given as left bottom width height

6.7.4 Power Render

A power render is used to show the power provided by a power source (commonly a robot’s batteries).

Example

```xml
<power power="Batteries" bounds="0.8 0.9 0.2 0.1"/>
```

Figure 6.11: Power render showing robot’s battery voltage.

Configuration

Attributes

invertX - whether or not to invert the X axis (default false)

invertY - whether or not to invert the Y axis (default false)

rotate - rotation in degrees to be applied to the render (default 0)

bounds - the bounding box for the render, given as fractions of the panel, given as left bottom width height

power - the name of the power device to display

name - the same as power

type - what value to print: charge, watts or percent

colour - the colour to draw the power
6.7.5 Tilt Render

The tilt render is used to show a robot’s roll and pitch. It shows a theoretical ground plane that extends to infinity.

Example

```xml
<tilt pos3d="RobotAtt" bounds="0.0 0.8 0.2 0.2"/>
```

Configuration

Attributes

invertX - whether or not to invert the X axis (default false)

invertY - whether or not to invert the Y axis (default false)

rotate - rotation in degrees to be applied to the render (default 0)

bounds - the bounding box for the render, given as fractions of the panel, given as left bottom width height

pos3d - the name of the Pos3D device to display

name - the same as pos3d

order - the rotation order to use to determine roll and pitch to display (default rpy)

colour - the colour to fill in the ground section of the render

6.7.6 Overlay Render

An overlay render is used to overlay a monochrome image (converted to mono 8bit) from one camera over the main camera. It uses the pose and field of view of both camera to position the overlay. It is possible to configure the overlay render to only overlay pixels in the overlayed image whose values are within a specified range.
Figure 6.13: Overlay of thermal image.

Example

```xml
<overlay camera="Thermal" min="200" max="255" toggle="t"
        minKey="r" maxKey="y" />
```

Configuration

Attributes

- `invertX` - whether or not to invert the X axis (default `false`)
- `invertY` - whether or not to invert the Y axis (default `false`)
- `rotate` - rotation in degrees to be applied to the render (default `0`)
- `bounds` - the bounding box for the render, given as fractions of the panel, given as `left bottom width height`
- `camera` - the name of the camera whose image is to be overlayed
- `name` - the same as `camera`
- `toggle` - key that will be used to toggle on/off the overlay display
- `min` - the minimum value of the camera’s image to be shown in the overlay (default `0`)
- `max` - the maximum value of the camera’s image to be shown in the overlay (default `255`)
- `minKey` - the key to be used to decrease the minimum displayed value, holding `Alt` will increase the minimum displayed value
- `maxKey` - the key to be used to increase the maximum displayed value, holding `Alt` will decrease the maximum displayed value
6.7.7 Preset Render

A preset render is used to display the current configuration of a robot (including joint positions). It can also be used to set a number of preset robot configurations that can be selected using the numerical keys on a keyboard. Holding Alt while selecting a preset will overwrite the preset configuration with the current robot configuration.

Figure 6.14: Current configuration of the robot and some available preset positions.

Example

```xml
<preset type="Emu" arm="Arm" bounds="0.0 0.0 1.0 0.2" >
  <preset value="0 0 0" />
  <preset value="45 0 0" />
  <preset value="-45 0 0" />
  <preset value="90 0 0" />
  <preset value="-90 0 0" />
</preset>
```

Configuration

Attributes

invertX - whether or not to invert the X axis (default false)

invertY - whether or not to invert the Y axis (default false)

rotate - rotation in degrees to be applied to the render (default 0)

bounds - the bounding box for the render, given as fractions of the panel, given as left bottom width height

type - the type of robot being used

- other attributes as defined by the type of robot

6.7.8 Target Render

Target renders are used to show the field of view for other cameras with respect to the main camera. Optionally a target render can be configured to execute a foreground task on the robot. When a target render is set to notify a targeting box is drawn within the camera’s field of view. This targeting box can be moved using the arrow keys.
Figure 6.15: Target render of SwissRanger field of view.

Example

```xml
<target camera="SwissRanger" key="v" task="CollectSnap" />
```

Configuration

Attributes

invertX - whether or not to invert the X axis (default false)
invertY - whether or not to invert the Y axis (default false)
rotate - rotation in degrees to be applied to the render (default 0)
bounds - the bounding box for the render, given as fractions of the panel, given as left bottom width height
camera - the name of the camera whose field of view should be shown
name - the same as camera
key - a key that will notify a foreground task that something of interest has been found
task - the name of a task to be notified of a located target
colour - the colour of the bounding box to use for indicating the camera’s field of view
Chapter 7

Devices

7.1 Player

To save the effort of writing a bunch of new drivers for all hardware CASRobot is able to connect to Player and so work with many devices for which Player drivers are already written.

7.1.1 Example

```xml
<robot name="Example">
    <player host="127.0.0.1">
        <camera name="Camera" id="0"/>
        <laser name="Laser" id="0"/>
        <pos1d name="Arm" id="0"/>
        <pos2d name="Motors" id="0"/>
        <pos3d name="Orientation" id="0"/>
        <power name="Batteries" id="0"/>
        <ptz name="PTZ" id="0"/>
    </player>
</robot>
```

7.1.2 Configuration

Attributes

- **host** - the name/ip address of the machine hosting player (default 127.0.0.1)
- **port** - the port player is listening on (default 6665)
- **update** - how many milliseconds we should aim at for receiving updates over Player (default 100)
- **required** - if true then if we can’t connect Player on robot start we throw an exception (default false)
- **log** - whether or not all client devices on this connection should be logging, each client can override this option (default false)
Children
camera - a camera player client, see section 7.1.3
laser - a laser player client, see section 7.1.4
pos1d - a pos1d player client, see section 7.1.5
pos2d - a pos2d player client, see section 7.1.6
pos3d - a pos3d player client, see section 7.1.7
power - a power player client, see section 7.1.8
ptz - a ptz player client, see section 7.1.9

7.1.3 Camera
CASRobot can talk to player cameras. Both RAW and JPEG compressed cameras are supported.

7.1.4 Laser
CASRobot can use lasers through player.

7.1.5 Pos1d
CASRobot can use player to talk to Pos1D devices.

7.1.6 Pos2d
CASRobot can use player to talk to Pos2D devices.

Configuration
carlike - some Pos2D devices need to be told their to be driven like a car, if so this attribute can be set (default false)

7.1.7 Pos3d
CASRobot can use player to talk to Pos3D devices.

Configuration
quaternion - our in house xSens driver actually uses quaternions, this attribute allows us to read in the quaternions and process them
quaternions - same as quaternion
embeddedJoints - some of our robots have an autoleveller connected to an xSens, this indicates that the joint positions for the servos in the autoleveller are included in the pos3D player data
7.1.8 Power
CASRobot can user player to talk to Power devices.

7.1.9 PTZ
CASRobot can user player to talk to PTZ devices.

7.2 Delogger
It is possible to create logs of sensor data that can then be replayed. This means that development and testing of some components can be done without requiring direct access to a robot.

7.2.1 Example

```xml
<delog log="mylog" skip="10000">
  <laser name="Laser"/>
</delog>
```

7.2.2 Configuration
Attributes

- log - which log to read from
- delay - how often to check if data needs to be updated in milliseconds (default 5)
- interval - same as delay
- skip - how many milliseconds in to the log to rapidly skip to (default 0)
- trace - whether or not to write extra log methods to trace a bug (default false)

Children
- camera - a camera device
- laser - a laser device
- pos1d - a pos1d device
- pos2d - a pos2d device
- pos3d - a pos3d device
- power - a power device
- ptz - a ptz device
7.3 Cameras

CASRobot is able to work with a number of cameras. These are some that come standard.

7.3.1 Fake

A fake camera isn’t a physical camera but rather a simple way for generated images to be sent to the GUI. If a robot configures a fake camera then tasks can get the fake camera to store an image and it will be streamed just as though it were a real camera. As far as the GUI is concerned there is no difference between a normal and a fake camera.

Example

```xml
<camera name="Stream" type="fake" />
```

Configuration

A fake camera is just a Streaming Camera and so uses the same configuration options.

7.3.2 CASCam

A CASCam is a subtype of camera that uses LibCAS to interface with the camera.

Example

```xml
<camera name="SwissRanger" type="CasCam" subtype="swissranger" id="0"
mode="combined" >
  <fov x="50" y="44" />
</camera>
```

Configuration

A CASCam is a subtype of Streaming Camera so it can accept all of its configuration options as well as:

Attributes

subtype - the type of camera to pass to LibCAS

- `dc1394` - a standard firewire camera
- `thermal` - a thermal firewire camera
- `swissranger` - a SwissRanger
- `unicap` - a camera require use of the unicap library
- `uvc` - a UVC based camera
- `v4l` - a camera that works with Video4Linux
- `ueye` - a uEye camera
id - an identifier for LibCAS to select the camera

invert - if the camera is upside down LibCAS can flip it (default false)

delay - how long to wait in milliseconds between image grabs (default false)

bayer - indicates the camera image is bayer formatted and which format it is (default none)

min - sets the minimum temperature for thermal camera to use in normalisation (default 25000)

max - sets the maximum temperature for thermal camera to use in normalisation, -1 means use maximum possible value (default 35000)

mode - sets the mode to use for a SwissRanger (default range)

  range - get range image
  amplitude - an intensity of reflection image
  combined - a range image and an intesity image, one above the other
  night - a night vision image (only available on some SwissRangers)

7.3.3 NightSwiss

A NightSwiss is a software camera that takes a combined image from a Swiss-Ranger and turns it into a night vision image.

![Side by side images from a SwissRanger and a NightSwiss camera.](image)

Figure 7.1: Side by side images from a SwissRanger and a NightSwiss camera.

Example

```
<camera name="NightView" type="NightSwiss" camera="SwissRanger" >
  <fov x="50" y="44" />
</camera>
```
A NightSwiss is a subtype of StreamingCamera (see section 3.3.1) so it can accept all of its configuration options as well as:

**Attributes**

- **camera** - the name of a camera from which to retrieve the combined image

### 7.4 Lasers

CASRobot usually uses Player for drivers to access lasers. But due to some inefficiencies in how Player talks to some devices an internal driver has been written for a Hokuyo UTM-30LX.

#### 7.4.1 UTM Laser

A driver has been written to allow CASRobot to talk directly to a Hokuyo UTM-30LX scanning laser range finder.

![Hokuyo UTM-30LX scanning laser range finder](image)

*Figure 7.2: A Hokuyo UTM-30LX scanning laser range finder.*

#### Example

```
<device type="utmlaser" port="/dev/ttyACM0" />
```

**Configuration**

**Attributes**

- **port** - the port which the laser is connected to (default `/dev/ttyACM0`)
- **log** - whether or not the laser scans should be logged (default `false`)
- **update** - desired update frequency in milliseconds (default 100)
asynchronous - whether to use asynchronous streaming from laser (default true)
Chapter 8

Tasks

This chapter describes some of the background and foreground tasks included in the standard CASRobot distribution.

8.1 Landmark/Victim Location

This section describes tasks for identifying and locating victims and landmarks.

8.1.1 CollectSnap

The CollectSnap foreground task tries to locate an object visible in the robot’s cameras. The object’s description, type, and status are determined by the arguments passed to its execute method. Included in the snap are the latest images from any configured cameras and a full colored point cloud.

Example

<foreground name="CollectSnap"
cameras="Wide Zoom Thermal SwissRanger"
/>

Configuration

Attributes

cameras - a list of the names of cameras whose latest images should be included in the snap

8.2 Position Tracking

This section describes the standard background tasks for determining the robot’s position.
8.2.1 MBPosTrack

MBPosstrack is a background task that matches the current laser scan against previous laser scans to estimate the current position of the robot. It can be configured to display the current local map.

![Local map used by MBPosTrack for localisation.](image)

Figure 8.1: Local map used by MBPosTrack for localisation.

Example

```xml
<background name="MBPosTrack" >
    <localmap
        occupancySafetyCells = "5"
        occupancyRadiusCells = "6"
        maxMoveAngleDeg = "30"
    />
</background>
```

Configuration

Attributes

- **showGraphics** - if 0 (default) then don’t show graphics, if >= 1 then show local map, latestScan and debug windows as configured

- **showLPS** - whether or not to show the local map (default true)
showCurrent - whether or not to show the latest laser scan (default false)

showDebug - whether or not to show a debug image showing cell convergence (default false)

maxIterations - the maximum number of ICP cycles to run, lower == less processing, but might not converge (default 200)

maxMoveX - a single laser scan can’t cause a motion larger than this, higher == more possibility for errors but also allows slower processing (default 0.8)

maxMoveY - a single laser scan can’t cause a motion larger than this, higher == more possibility for errors but also allows slower processing (default 0.8)

maxMoveAngleDeg - a single laser scan can’t cause a motion larger than this, higher == more possibility for errors but also allows slower processing (default 40)

mapSizeMeters - the size of the map in meters (default 10.0)

coarseErrX - when change from a single ICP iteration is below this, coarse algorithm has converged (default 0.001)

coarseErrY - when change from a single ICP iteration is below this, coarse algorithm has converged (default 0.001)

coarseErrThDeg - when change from a single ICP iteration is below this, coarse algorithm has converged (default 0.01)

errX - when change from a single ICP iteration is below this, fine algorithm has converged (default 0.001)

errY - when change from a single ICP iteration is below this, fine algorithm has converged (default 0.001)

errThDeg - when change from a single ICP iteration is below this, fine algorithm has converged (default 0.1)

sleepUsecs - sleep this much between cycles, saves processing power, but since algorithm waits for new laser readings it could cause some to be missed (default 5000)

resetHDist - recalculate heuristic values every resetHDist meters (default 2.0)

maxPriorDist - if the previous update moved or turned more than this, don’t use it as the default motion for this update (default 0.1)

maxPriorAngleDeg - if the previous update moved or turned more than this, don’t use it as the default motion for this update (default 20)

echoDriver - this is the name of a position3d_echo driver so that position updates can be sent back to player

useOccupancyGridMBICP - if true (default) use the occupancy grid algorithm in this file otherwise use the standard MBICP in MBPosTrackStd
useMotionReset - if true assume motion is zero and then use MBICP, otherwise start with motion == to last motion, as long as it's not too big (default false)

Children

localmap - configuration options for grid based MBICP, these config options are processed by the local map (see section 8.3.1)

mbicp - configuration options for standard MBICP

\begin{verbatim}
std_max_laser_range - ???
std_angular_window - ???
std_radial_window - ???
std_L - ???
std_laserStep - ???
std_MdistInter - ???
std_filter - ???
std_ProjectionFilter - ???
std_AsocError - ???
std_MaxIter - ???
std_errorRatio - ???
std_errx_out - ???
std_erry_out - ???
std_errt_out - ???
std_IterSmoothConv - ???
std_Exclusion_xmin - ???
std_Exclusion_xmax - ???
std_Exclusion_ymin - ???
std_Exclusion_ymax - ???
\end{verbatim}

8.2.2 ICPPosTrack

ICPPosTrack is a background task periodically grabs a point cloud and sends it to robot listeners such as the GUI. It can be configured to display data from the position tracker and the corrected robot path.

Example

\begin{verbatim}
<background name="ICPPosTrack" />
\end{verbatim}
Configuration
Attributes

showGraphics - if 0 (default) don’t show graphics, if >= 1 then update correspond window

logIntervalMs - number of milliseconds between updates (default 100)

filter - if true then invalid points are filtered from the point cloud to save bandwidth

filterColours - if false (default) then coloured points from combining a SwissRanger and a colour camera are added to the point cloud, when filtering is enable the coloured points are also filtered, if true all coloured points are discarded

colourPointSkip - to save bandwidth only 1 in colourPointSkip coloured points are added to the cloud (default 4)

8.3 Autonomy
TODO: ... 

8.3.1 LocalMap
TODO: ...

Configuration
Attributes

laserSkip - speedup parameter, every laserSkip laser values used in fine alignment (default 5)

1 - L value from MBICP, inf means standard ICP, towards zero means rotation more important, primary method for tuning MBICP (default 2.0)

occupancyRadiusCells - radius in cells around occupied cells, more == safer but closes off narrow passages (default 7)

occupancySafetyCells - threshold around occupied radius cells, higher == try to travel further from wall (default 3)

maxObservationCount - maximum number of observations, affects occupancy probability of cell (default 1000)

maxLifeCount - when a cell is observed its life is set to this after it is unobserved this many scans it is removed (default 1000)

maxAlignmentDist - laser readings must be closer than this to be matched in MBICP (default 0.6)

MaxSegLen - adjacent cells must be closer than this to be considered a segment (default 0.1)
coarseAlignmentSizeCells - radius in cells to search for coarse alignment, increasing this increases processing time but also permits larger motions between scans (default 1)

coarseAlignmentDFix - scale parameter for distance in coarse alignment, it should be greater than the max laser distance (default 7.0)

courseAlignmentCurrentWeight - weight of current cells in coarse alignment, currently this part of the algorithm is disabled (default 0.5)

alignmentCellSize - radius in cells to search for fine alignment (default 3)

alignmentCurrentWeight - weight of current cells in fine alignment (default 1.0)

alignmentUnobservedWeight - weight of non-current cells in fine alignment (default 1.0)

alignmentDFix - scale parameter for distance in fine alignment, it should be greater than the max laser distance (default 6.0)

minNumberObservations - not currently used (default 10)

minObservationsOfObstacle - only after this many scans observe an object does it get added to the map, higher = less transients but more likely to hit something before it appears (default 75)

negObsOfObstacle - only after this many non-observations of an object is it made old in the map, higher = objects persist for longer, but might avoid things that aren’t there any more (default 100)

depthVarianceThreshold - ??? (default 0.3)

depthOccupancyThresholdAdjustment - ??? (default -3)

8.3.2 BasicController

Basic controller for the robot, moves the robot directly to a point based on position tracking. It doesn’t consider any obstacles. Holds a reference to a planner task to make decisions.

Example

```xml
<background name="BasicController" speed="0.20" turnrate="0.5"
SlowTurnThresholdDeg="10.0"
OnlyTurnThresholdDeg="25.0"
>
<planner
MinObservationsOfObstacle="3"
NegObsOfObstacle="50"
SendGraphics="1"
UseDepthCamera="0"
>
DepthVarianceThreshold="0.005"
</background>

Configuration

Attributes

speed - the robot’s maximum forward speed

turnrate - the robot’s maximum rate of turn

delay - when autonomously recovering from radio dropouts this is how long to wait in milliseconds before deciding that a dropout has occurred

reachedDestinationThreshold - when the robot is this close to a point it has reached it (default 0.2)

reachedAngleThresholdDeg - when the robot is within this angle of a destination it’s reached it (default 5.0)

minimumAngleWeight - the minimum fraction of the turn rate that the robot can do (default 0.3)

slowTurnThresholdDeg - if the robot is further than this angle from facing its target it drives at only a percentage of its speed and turns at full speed (default 20.0)

onlyTurnThresholdDeg - if the robot is further than this angle from facing its target it turns at full speed and does not move forward (default 30.0)

maxSpeedDistanceThreshold - if the robot is closer to its destination than this it only moves at a percentage of its maximum speed (default 1.0)

Children

planner - configuration options for the controller’s local planner (see section 8.3.3)

8.3.3 LocalPlanner

Autonomous controller, makes motion decisions based on local map. Implements simple autonomy, wall following, bump and go, straight line. Although it is a background task LocalPlanner should be instantiated as a child of BasicController and not as a separate task.

Configuration

Attributes

speed - the robot’s maximum forward speed

turnrate - the robot’s maximum rate of turn

delay - when autonomously recovering from radio dropouts this is how long to wait in milliseconds before deciding that a dropout has occurred
raytraceStepSize - ray makes steps of this size, depends on size of cells (default 0.01)

avoidHistoryPath - whether or not to avoid crossing robot’s path (default false)

useDepthCamera - whether or not to use a depth camera to determine if a cell is traversible (default false)

showGraphics - whether or not to display the traversibility map (default false)

sendGraphics - whether or not to send the map to the GUI using a streaming camera called LocalPlannerCam (default true)

displaySizeFactor - image uses this many pixels (in each dimension) for each cell (default 2)

showScans - if showGraphics and showScans are both true then the scans are shown in the scans window (default false)

sleepUsecs - sleep this many usecs between cycles, planner doesn’t have to run that fast (default 1000000)

startBehaviour - initial autonomous behaviour

0 - do nothing (default)
1 - follow left wall
2 - follow right wall
3 - bump and go left
4 - bump and go right

Children

localmap - configuration options for grid based MBICP, these config options are processed by the local map (see section 8.3.1)

wallfollow - options wall following

WallFollowCheckAngleDeg - ??? (default 100.0)
WallFollowStartDeg - ??? (default 85.0)
WallFollowSearchDefaultDeg - ??? (default 95.0)
WallFollowSearchDeg - ??? (default 85.0)
WallFollowSearchRatio - ??? (default 1.0)
WallDistCheckAngleDeg - ??? (default 115.0)
WallAngleAdaptCount - ??? (default 2)
WallMaxVSearchCount - ??? (default 100)
WallFollowEndDeg - ??? (default -270.0)
WallFollowIncDeg - ??? (default 1.0)
WallTurnEscapeDist - ??? (default 0.2)
MinWallFollowDist - ??? (default 0.5)
GoodWallFollowDist - ??? (default 1.0)
WallSearchDist - ??? (default 1.0)
WallFollowSearchAngDeg - ??? (default 20.0)
VoronoiThresholdScore - ??? (default 6.0)
EmptyCellScore - ??? (default 5.0)
VoronoiHorizonScore - ??? (default 1.0)

straightpath - options for driving straight

AngDeviationDuringForwardDeg - ??? (default 20.0)
ForwardBehaviourSearchDist - ??? (default 1.0)
ForwardMinDist - ??? (default 0.3)
Chapter 9

Mapping

TODO: ...

9.1 GeoTIFF

TODO: ...

Figure 9.1: GeoTIFF image of an autonomous run made by [Emu] during the Thailand Open 2009.

9.2 FastSLAM

TODO: ...
9.2.1 Example

```xml
<map name="Map" type="FastSLAM" robot="Pioneer"
     particles="150"
>
</map>
```

9.2.2 Configuration

Attributes

- **particles** - the number of particles to use (default 200)
- **log** - whether or not to save the map as it changes (default **true**)
- **delay** - only save map every N milliseconds (default 0)
- **interval** - same as **delay**
- **robot** - the name of a robot that will be providing point clouds for map updates
- **task** - the name of an autonomy task to be resumed on hidden snaps
- **gain** - the higher the gain the more quickly the particles converge to a single location (default 0.0003)
- **initialCellProb** - the initial probability of any cell being occupied (default 0.5)
victimThreshold - the distance in meters between detected snaps at which they're considered a new landmark/victim (default 1.0)
distanceThreshold - the distance in meters a robot will travel to force an update (default 0.2)
angularThreshold - the angle in degrees a robot will turn to force an update (default 15.0)
heightThreshold - height in meters at which a cell must be occupied to be considered occupied (default -10.0)
useDepthCamera - whether or not to ignore coloured points (default true)
laserAngleOfObservationsDeg - alternative to depth camera, laser rays in this angle from head are considered searched (default 20.0)
depthPointSkip - only use every Nth point from depth camera (default 8)
useFeatures - use feature detector, Don’t Change! (default 0)
performMapUpdates - whether or not to update cell probabilities (default true)
addMotionCopy - whether to on a resample add a copy of the motion particle (default true)

Children

k - changes k parameters
  range - (default (5.0 * 1.0 / 100.0))
  theta - (default (1.0 * (\pi / 60.0) / (2.0 * \pi)))
  drift - (default ((\pi / 90.0) / 0.20))
  shift - (default (1.0 / (2.0 * \pi)))

smear - changes smearing parameters
  x - maximum smearing in X dimension in meters (default 0.1)
  y - maximum smearing in Y dimension in meters (default 0.1)
  z - maximum smearing in Z dimension in meters (default 0.05)
  yaw - maximum smearing in YAW rotation in degrees (default 6.0)

initialPose - changes smearing parameters
  x - initial X position of robot (default 0)
  y - initial Y position of robot (default 0)
  yaw - initial YAW of robot (default 0)
Chapter 10

Libraries

Both RobotServ and RobotGUI are able to loaded external libraries that can be used to extend their capabilities. This chapter describes what extensions can be added through libraries and how to go about building them.

10.1 Cameras

TODO: ...

10.2 Devices

TODO: ...

10.3 Tasks

TODO: ...

10.4 Maps

TODO: ...

10.5 Panels

TODO: ...

10.6 Renders

TODO: ...
10.7 Example

There is an example of how to build a library for use with RobotServ and RobotGUI. To check it out run:

```
svn co http://robolab.cse.unsw.edu.au/svn/rescue/trunk/casrobot-examplelib
```

If RobotServ and RobotGUI are properly installed you should now be able to compile and run it.
Chapter 11

Robots

There are a number of robot that CASRobot has been used with. This section give a brief overview of some that it is currently being used on.

11.1 Negotiator

TODO: …

Figure 11.1: Negotiator, who came 2nd in the Mobility class in the 2009 RoboCup Rescue Robot League competition.

Ribbons

2009
• Finalist
  - 2nd RoboCup Rescue Robot League: Best-in Class Mobility
  - RoboCup Rescue Robot League: Innovative Operator Interfaces Award

2007
• Finalist

11.2 Emu

TODO: …
Figure 11.2: Emu, who came 1st in the Autonomy class in the 2009 RoboCup Rescue Robot League competition.

Ribbons

2009  • Finalist
   🎁 1st RoboCup Rescue Robot League: Best-in Class Autonomy
   🎉  RoboCup Rescue Robot League: Innovative Operator Interfaces Award

2007  • Finalist

11.3 Pioneers

TODO: ...

Figure 11.3: Flinders and Wills resting after a big day.

11.4 Pioneer AT

TODO: ...
11.5 Peoplebot

TODO: ...

Figure 11.5: PeopleBot awaiting instructions.

11.6 Weeder

TODO: ...

11.7 Caster\[1\]

TODO: ...

66
Figure 11.6: CASTER, the 2005 Team CASualty robot, which came 3rd in the 2005 RoboCup Rescue Robot League competition.

Ribbons

2006
- Semi-Finalist
  - 2nd RoboCup Rescue Robot League: Best-in Class Autonomy

2005
- 3rd RoboCup Rescue Robot League

11.8 Truxar
TODO: ...

11.9 Packbot
TODO: ...
Appendix A

Frames of Reference

A.1 Robot Coordinates

Figure A.1: The reference frame for robot coordinates.
A.2 World Coordinates

![World Coordinates Diagram](image)

Figure A.2: The reference frame for world/map coordinates.

A.3 Player Coordinates

**TODO:** ...

A.4 OpenGL Coordinates

**TODO:** …
Bibliography

