Design Specification

Group 3

September 20, 2012

Project Identity

Group 3, 2012/HT, "The Robot Dog"

Linköping University, ISY

Name	Responsibility	Phone number	E-mail
Martin Danelljan	Design	072-372 6364	marda097@student.liu.se
Marcus Eriksson	PR	073-647 7180	marer014@student.liu.se
Daniel Hultqvist	Project manager	070-289 2859	danhu635@student.liu.se
Victor Johansson	Test	072-500 8006	vicjo046@student.liu.se
Johannes Markström	Documents	070-353 9655	johma564@student.liu.se
Niklas Pettersson	Quality	076-634 2903	nikpe872@student.liu.se

Customer: Michael Felsberg, michael.felsberg@liu.se, CVL, LiU
Course examiner: Vasileios Zografos, vasileios.zografos@liu.se, CVL, LiU
Supervisor: Liam Ellis, liam.ellis@liu.se, CVL, LiU

Contents

Document history

Version	Date	Changes	Made by	Reviewed
0.1	2012-09-17	First version	Johannes Markström	2012-09-19
0.2	2012-09-20	Second version	Johannes Markström	

1 Introduction

This document specifies the design of the robot and the software used to control it. Each software module is described in its own section. The document will be updated as the project is progressing, especially when a design decision has been made during a sprint. Specific methods referenced to a paper describing it, once it has been implemented and tested more specific details will be filled into this document.

2 System overview

In the basic version, the robot will have two different states. When the system is started or asked to learn a new operator, the system will enter the "learning" state. In this state, the robot will be stationary and learn the appearance of the person in front of it. When the robot is ready to go it will enter the "following" state, where it will try to follow the operator. Some online learning of the operator's appearance in the following state could also be considered.

Our system will be divided into the areas given by the block diagram in figure 1. The image capture will capture images from the camera and do the necessary conversion so that the images can be fed to the person detector and the collision detection.

The person detector will be used to detect persons in front of the robot. The resulting bounding boxes around the detected persons will be used by the operator classifier. When the robot is in the learning state, these boxes will be used to identify the part of the image that contain the operator. While in the following state, the operator classifier will calculate values for each bounding box that measure the probability of the bounding box containing the operator.

The information from the operator classifier will be fused with the information on the earlier positions of the operator in the image, to decide which bounding box contains the operator.

The relative distance and angle from the robot to the operator will then

be measured, using the identified bounding box. The measured values are filtered and fed to the steering system. The steering system uses the relative position of the operator to calculate the steering commands (including speed). The steering system also uses information from the obstacle detection, so that collisions can be avoided.

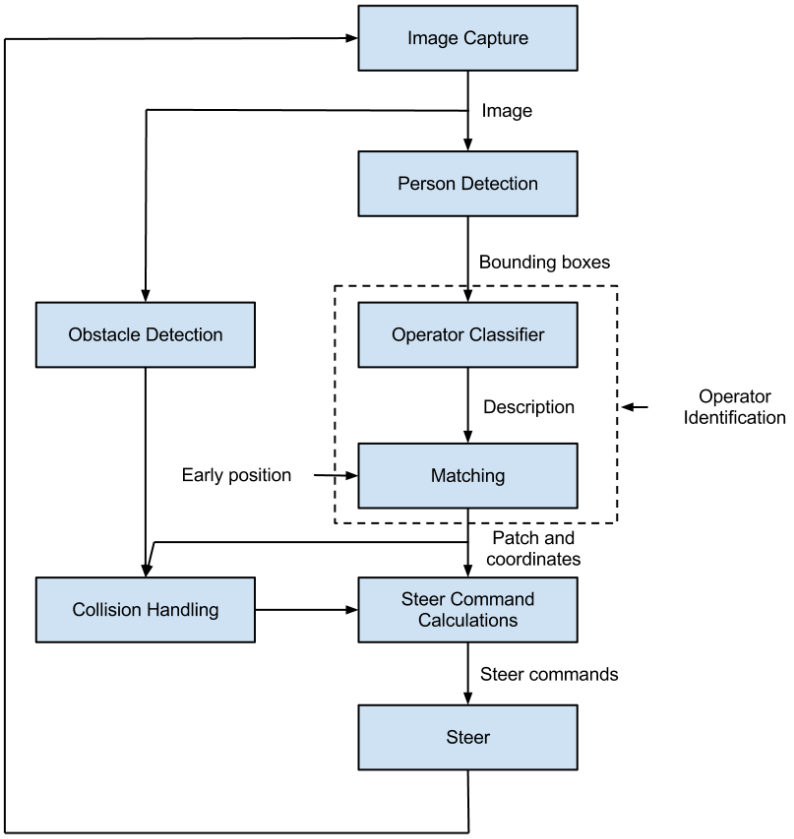

Figure 1: Overview of the system.

3 Image Capture

The robot has a mounted Point Grey color camera which we intend to use for the tracking and detection of the operator. There already exist an image capture module that uses Point Grey API for capturing images. This module is used with the gray scale Point Grey camera.

Due to the fact that we have upgraded the camera to a color camera we need to update the Point Grey image matrix - OpenCV image matrix converter to include the color space. The image capture module takes an image and stores it in a thread-safe buffer so that any other module running on a separate thread can reach the buffer anytime.

4 Person Detection

The first step in locating the operator is to first find all persons that is present in the current image. There are many available methods for object/pedestrian detection. This project will firstly look into three approaches *HOG* [1], *Chamfer system* and *Hough forest*. Which method that will be used depends on how easy it's to implement given that it is robust and fast enough.

4.1 HOG method

The basics of *HOG* is that an area is divided into smaller regions called cells. In each cell a histogram of gradient directions is computed. The combination of all the histograms is the descriptor of the area. The *HOG* pedestrian detector works by finding areas in an image where its *HOG* descriptor matches one of a human. Pedestrian detection using *HOG* is already implemented in *OpenCV* [2] and will be the one that first is going to be tested.

4.2 Chamfer method

Chamfer is a template matching system. An edge image is extracted from the input image. From the edge image a distance transformation is made where the pixels are given the value of the distance to the closest edge. The distance image is then correlated with the template and if there are values under a certain threshold it is said to be a match. Normally there is more

than one template and the system can be pretrained by making a database of contours that is interesting (in this case contours of humans). The database can be made up as a tree-structure where one subtree is more similar than the rest of the tree. The advantage of this is that you don't need to do matching with all templates which will lead to some speedup.The *chamfer system* is described more in detail at the gravila website [3].

4.3 Hough Forest

The basic idea is to create a tree from test data were every tree represent an individual class. The tree is constructed from patches consisting of pieces of the class (positive) and non class (negative) and distances to the centroid of the object it represents. Each node represent either a positive or negative feature. Each node vote for or against the test patch being in the class or not. Follow the branch that gives the best match, the end results in a probability of the test patch being in the class or not and gives a location of the centroid of the object. This is described more in detail in an article [4] and the source code can be downloaded at the vision web page [5]. The source code has some extensions compared to the theory.

5 Operator Identification

This section describes methods we might use to identify the operator. All methods will not necessarily be implemented if one implemented method already gives good performance.

5.1 Overview

These classifiers will be used to distinguish different people, namely the operator from non operators. We will base the classifier on color histograms feature descriptors and then use one of three versions. In theory, the operator classifier is only needed for requirements with prio 2 or higher, however the performance might be enhanced for the basic requirements.

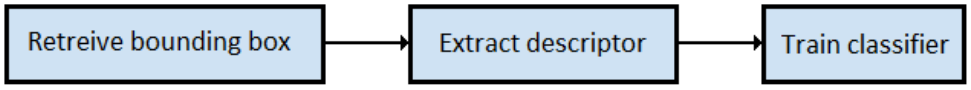

Figure 2: Work flow for the learning phase.

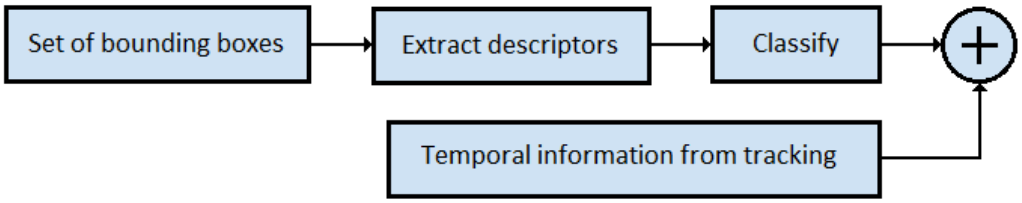

Figure 3: Work flow for the classification.

5.2 Descriptor

The patch will be subdivided into smaller patches, on which color histograms will be calculated. The color histograms will be the base of the descriptors. The different areas will have different weights, where the center parts have higher weights, i.e. more influence.

5.3 Classification

We will consider three different methods for classifying the operator. If one method has good enough performance, the others might not be implemented.

5.3.1 Histogram distance

The first method will be based on simply the histogram distance compared to a database, containing learned histograms of the operator. If a match is good enough, it is considered true. If it is pretty close, but not a full match, it will be stored as a new feature vector describing the same object, as it is probably a result of the operator changing direction or posture.

5.3.2 Neural Network

There is already a module that uses neural networks for learning on the robot which we can test. A neural network should be quite easy to implement with real-time performance, if the available code is not sufficient.

5.3.3 Hough forest

In this case data and information about the operator has to be learned before the robot can be deployed. However, the Hough forest is a very fast and stable solution, which could prove superior to the more simple neural networks method. Another main advantage of using Hough forest is that we can also use it for the detector and even go so far as to integrate the detector and descriptor into one module.

5.4 Tracking

As an additional tool to classify the operator, the movement of the operator should be tracked. This can be done, for instance, by KLT tracking. An other option is to use a Kalman filter to predict the trajectory of the path.

5.5 Matching

It is assumed that between two consecutive frames with little time difference the relative motion of the operator will be small. Therefore a simple matching algorithm is to just compare the distance between the centroids of the candidate persons with the operator position in the last frame.

This approach will probably work well under some conditions, but is limited by the detectors ability to correctly detect the operator. Problems can arise if the detector is giving false positives, or having two or more persons on approximately the same distance from the last position of the operator, in the robots field of vision.

To make the matching more robust and able to handle more than one moving object, the description of the appearance of the candidate persons will also be compared with the learned operator. The output will be a probability of the candidate persons matching the operator based on appearance and will be weighted with the information on difference in relative position.

6 Obstacle Detection

For the robot to be practically useful it needs to be able to follow its operator without crashing into objects in its path. In this project it's assumed that the deployment environment has no obstacles, so the challenge is to avoid driving into a wall or the operator.

There is already an existing module for obstacle detection present in the framework which makes a 3d-reconstruction from motion and estimates the dominant plane. This is then assumed to be the ground plane. A machine learning approach to find out the depth of an image is to classify each image against a database and then predict the depth to be similar to that of a similar image. Another interesting method is [6] which uses optical flow to detect obstacles.

Our primary intention is to use the system already present on the platform.

7 Collision Handling

If an obstacle is found the robot will have to take action to avoid a collision. A simple way of handling this is the *Crash and turn algorithm* which works as follows:

1. Move in the direction of your target.
2. If you hit a wall, turn in the direction that puts you closest to the target. If no choice is obviously better, pick one at random.

Since we don't want to hit a wall the obstacle detection have to warn in good time before an impact. This method is described more detailed in [7].

8 Steering Commands

This section describes the different ways the robot can be controlled and how the different methods work.

8.1 Calculations

The detected operators position and size in the image space will be mapped to a distance and an angle in the world space. The distance will be calculated from the height of the operators bounding box and the angle will be calculated using the horizontal distance between the operators centroid in the image and the camera center. The transformations will be done either by using the pinhole camera model or an interpolated curve from tabulated values. To make the steering more smooth and robust to noise, these values will be filtered with for example Kalman filters.

The steering and speed commands will be calculated by a controller, using a feedback control loop. The filtered values of the distance and angle relative to the operator are used as input, together with the user defined reference value for the distance. The controller could be a simple PD- or PID-controller.

When there's risk for collision the speed and turning will be overridden by the collision handler, see section 7.

8.2 Steering

We will be using a separate thread to control the robots actions. This module is already available but needs to be modified to suit our needs. The steering module will have three basic modes of operation:

- Completely automatic. The control unit receives control signals from a separate thread, telling the robot what to do.
- Semi-automatic. The robot receives control signals telling the robot about where and how to move, however the robot speed is controlled by a remote control. Using this method, it is easy to make sure that the robot does not crash.
- Manual. The robot is controlled by a remote control. Other modules might still be running and performing tasks. This mode could be used to test the obstacle detection system, if a potential collision is detected it will force the robot to stop.

9 Gesture Detection

When the robot is standing still(static scene) it will be able to recognize gestures that the operator might do. As a first approach we will look for the hands of the operator by looking at the sides of the bounding box of the operator. The hands are detected by using a color descriptor and the size of the hands can be calculated from the size of the operator's bounding box. The motion will be expressed as vectors that will tell the direction of the movement. If the movement exceeds a defined threshold we regard it as a gesture. The threshold will be independent of the distance from the robot to the operator by the mapping from the size of the operator.

By combining movements from right and left hand in y- and x-direction we can easily define some basic gestures/commands. The gestures will be defined later as this is a requirement with a lower priority.

10 Testing and Evaluation

In order to determine if a module and the system as a whole works according to expectations, some systematic testing and evaluation techniques will be defined.

10.1 Robustness

Here follows a preliminary description of the different testing techniques for determining the robustness of the system.

10.1.1 Steering

- Evaluate how well the robot steer towards the goal.

  One method to test this is by starting the robot in different orientations compared to the target. One can then measure how long time it takes for the robot to align with the target, as well as how much angular divergence there is when the robot is done aligning.

- Evaluate how well the robot keeps the distance.

One method is to simply start at different distances from the target and measure how long time it takes for the robot to enter the specified following distance, and how much it diverges when done.

10.1.2 Collision

- Evaluate how well the robot detects walls.

  One way to test this is to record a video while the robot is manually controlled in an empty room. This video could be used to calculate a receiver operating characteristic curve.

- Evaluate how well the robot avoids walls.

  This could be tested by having a couple of predefined speeds and angles which the robot will be started in relative to a wall. One should also define a goal position, on the other side of the wall. Then one could measure how often it fails to avoid, and how long time it takes for the robot to avoid the wall.

10.1.3 Detection

- Evaluate how well the robot detects persons.

  One way to test this is to record a video while the robot is manually controlled in a room with one, two and four persons. This video could be used to calculate a receiver operating characteristic curve.

10.1.4 Matching and classifier

- Evaluate how well the robot matches the same people.

  This could be tested on the same video as the detector, but with the different people marked and separated from the background manually. One can then measure how often the people are correctly matched between frames, how many mismatches there are, as well as how many people there are that the matching fails to match.

10.1.5 Gesture recognition

- Evaluate how well the robots recognize different gestures.

This could be tested by recording a video of different people doing different gestures multiple times. The gesture recognition could be performed on this video, and then it would be possible to count the number of correctly identified gestures, number of wrongly classified gesture as well as the number of missed gestures.

10.2 Speed

Our system must work in real-time so the speed of the different modules are very important. Every module should be clocked, so that modules that are too slow may be optimized for speed. As some modules are more important than others a priority list would look like:

1. Obstacle Avoidance
2. Person Detection
3. Matching
4. Steering Commands Calculations
5. Operator Classifier
6. Gesture Recognition

The first ones has higher priority and will be optimized first.

11 References

[1] Navneet Dalal, Bill Triggs, *Histograms of Oriented Gradients for Human Detection*. `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1467360`

[2] OpenCV crew, *Object detection*. `http://docs.opencv.org/modules/gpu/doc/object_detection.html`

[3] Gavrila, *The Chamfer system*. `http://http://www.gavrila.net/Research/Chamfer_System/chamfer_system.html`

[4] Juergen Gall, Victor Lempitsky, *Class-Specic Hough Forests for Object Detection*. http://www.vision.ee.ethz.ch/~gallju/download/jgall_houghforest_cvpr09.pdfl, 2009.

[5] Juergen Gall, Victor Lempitsky, *Class-Specic Hough Forests for Object Detection homepage*. http://www.vision.ee.ethz.ch/~gallju/projects/houghforest/index.html

[6] Toby Low, Gordon Wyeth, *Obstacle Detection using Optical Flow*. http://www.araa.asn.au/acra/acra2005/papers/low.pdf

[7] Ingemar Ragnemalm *Polygons feel no pain, So how can we make them scream?, Volume 2* 2012