Vision-based Velocity Control on a Philips Experimental Robot Arm

M.Sc. Thesis for Embedded Systems
by

Huub van Niekerk

October 3, 2014

Nulia tenaci invia est via
For the tenacious, no road is impassable

Contents

Abstract

The challenge in this thesis is to find out if an off-the-shelf embedded system can replace an off-the-shelf laptop or desktop computer when its task is to perform vision-based velocity control using inverse kinematics on a robotic arm.

The results of this thesis are that an algorithm was developed which had to be tested in simulation and should run (semi-)autonomously on an embedded system but there are no good test results on the algorithm.

Developing and testing an algorithm using an existing simulation proves to be very problematic as the used simulation software is very complex and has gone out of support by its developers.

Although the embedded system was chosen because it is equiped with a digital signal processor, I sadly found out that its proprietary driver is mutually exclusive with robot-messaging middleware, when it comes to operating systems´ kernel support: the choice was between the driver by using an old kernel or the middleware by using a new kernel. The latter was chosen.

A real-time software kernelpatch necessary to communicate with the robotic arm unfortunately was still in development in the final stage of this work.

Porting an inverse kinematics algorithm from Matlab to C++ and adapting the trajectory generating algorithm for middleware went well, but could not be tested thoroughly because of simulation and real-time issues. This also holds for testing the velocity control algorithm.

The conclusion of this report is that there is future work necessary in order to see if the developed algorithm for vision-based velocity control actually works.

1 Introduction

1.1 Robotic arms

Robotic arms are in use, amongst others, for medical and industrial purposes. For medical purposes, one can think of an arm prosthesis or a mind-controlled arm for paralyzed people. Industrial examples are automated welding or paint robot arms in car manufacturing.

Philips Applied Technologies has developed the Philips Experimental Robot Arm or PERA. The PERA is a compliant anthropomorphic arm which means it is a flexible arm which is shaped as a humanoid arm with similar kinematics. Philips states in its manual that they developed it as a research tool. As such, it is to be used only in a laboratory for the research of humanoid robotics because of its human kinematics. However, it is used outside laboratories as well: e.g. the Amigo-team of the Eindhoven University of Technology (TUe) uses the PERA for their robot. Amigo is an acronym for Autonomous Mate for IntelliGent Operations.

The PERA has clearly defined limitations in speed as well as in lifting weight. The arm is originally controlled by a computer running Ubuntu Linux as shown in Fig. 1 and the power controller as shown in Fig. 2.

Figure 1: Computer belonging to the PERA

Generally, the control of robots and robotic arms is taken care of by either off-the-shelf or industrial computers. In challenges such as the RoboCup [1], this is mostly not different. Even when the robotic device is remote-controlled, it is still controlled by a computer.

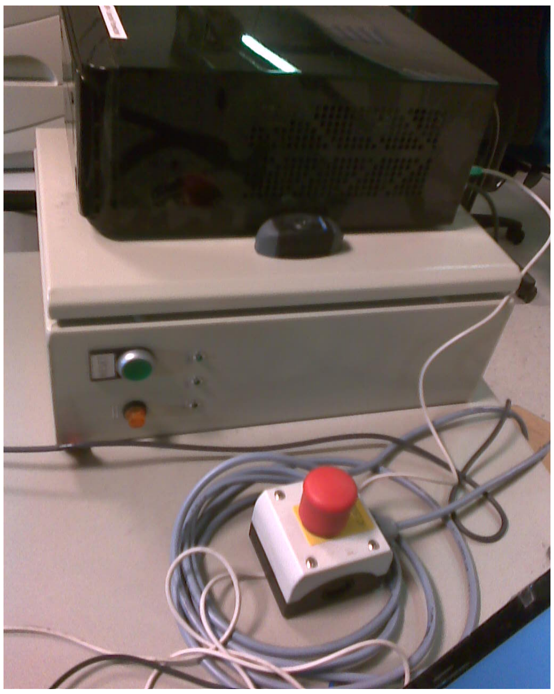


Figure 2: Original computer and Power controller with red Emergency stop button

At the Delft Biorobotic Laboratory, since January 2013 part of the Delft Robotic Institute, robots are mostly controlled by an off-the-shelf computer which is onboard the robot. For this thesis, the challenge is to see if an off-the-shelf general purpose computer is indeed necessary to control a robotic arm such as the PERA or if it can be done by an embedded system.

Vision is a common part of robotics by which objects can for example be detected or classified. Detection can for instance be done by using the interruption of a lightbeam or, more complex, processing an image from one or more cameras.

Velocity control can be necessary to prevent damage to either a robot itself or objects near a robot. For instance, in a car factory the welding or painting arm needs to be able to adjust its speed. That way it can keep up with the cars on the conveyor belt and it is able to work on the correct parts and avoid collision with a car. In case of the PERA, its motors have a maximum safe speed. If this speed exceeds this maximum, the speed might give vise to an over-voltage when decelerating which can damage the controller boards. Also, when an object is detected in its path and an alternative path is not known, the velocity has to be reduced.

The challenge for this thesis is to develop an algorithm for vision-based velocity-control with trajectory generation for the PERA. Because this thesis is about Embedded Systems, the challenge is reformulated to develop the algorithm and run it (semi-)autonomously on an off-the-shelf embedded system.

1.2 Related work

Baturone et all [2] describes an efficient design methodology allowing to start with any kind of fuzzy controller and then transforming it until a system suitable for easy DSP implementation is obtained. In Yang et all [3] an embedded fuzzy controller for a nonholonomic mobile robot is developed. The robot was built based on behaviour-based artificial intelligence, where several levels of competences and behaviours are implemented. A class of fuzzy control laws is formulated using Lyapunov's direct method. In Gohiya et all [4] a mobile robotic system for monitoring plants and environment parameters using a wireless network is envisioned and developed based on an ARM9 platform running Linux. Rinner et all [5] presents a rapid prototyping environment for flexible embedded systems on multi-DSP architectures. This prototyping environment automatically maps and schedules an application onto a multi-DSP architecture and introduces a special, lightweight reconfiguration environment onto the target platform. Chen et all [6] presents a new idea of constructing an architecture with DSP basement and supplementing it with some RISC features. It can exert the advantages of DSP architecture by instruction level parallellization and powerful memory access capability.

According to Chaumette and Hutchinson [7], vision can be used to locate the destination and with this knowledge minimize the error in movement. In this paper, the case is considered of a fixed goal pose in combination with a motionless target. This correlates to inverse kinematics. In HuChang et all [8] an embedded FPGA-based visual servoing platform (VSP) is proposed to meet the requirements on data transfer and computation capability for quick response visual servoing tasks. Do Hyoung et all [9] introduces an active vision system, which has a fast and simple system architecture that uses a high-speed serial communication bus. Litzenberger et all [10] presents an embedded vision system for object tracking applications based on a 128×128 pixel CMOS temporal contrast vision sensor.

In Zheng et all [11], two simple on-line smooth trajectory generators are developed by adapting available non-linear tracking differentiators using the known velocities of the given rough trajectory. Roel Pieters [12] developed Direct Trajectory Generation, or DTG, which deals with object avoidance. This can be achieved both online and offline. Torsten Kröger [13] developed a method for Online Trajectory Generation. Delsart et all [14] presents Tiji, a trajectory generation scheme, which is an algorithm that computes a feasible trajectory between a start and a goal state, for a given robotic system. It is geared towards complex dynamic systems subject to differential constraints. Chi-Kin Lai et all [15] addresses the problem of collision avoidance with moving obstacles for unmanned aerial vehicles.

1.3 Outline

This thesis applies Roel Pieters' algorithm to the PERA on an embedded system. As such, in section 2, robot modelling is descibed with respect to the PERA. Section 3 discusses the architecture of the embedded system software, and section 4 the implementation, after which section 5 describes the chosen embedded system. In section 6, the simulations that are done are discussed and section 7 discusses the experiments. In section 8 conclusions will be drawn followed by section 9 with recommendations and section 10 with suggestions for future work.

2 Robot Modelling

2.1 Introduction

When an existing robot is used to analyse the performance of a new algorithm, the robot has to be modelled in order to see if it is able to execute that algorithm. This model should physically and kinematically be similar to the real robot as much as possible.

As mentioned in the Introduction, the PERA is used in the Amigo robot for its left and right arm, which is modelled using the Gazebo [16] simulation environment. In order to not "reinvent the wheel", I decided to use the Amigo simulation for this work as well. In the DBL, only the right-side PERA is available, so only this part of the simulation was used.

To represent the model in Gazebo, the Unified Robot Description Format (URDF) representation is used. URDF is an Extensible Markup Language (XML) specification in which a robot is described in the Robot Operating System (ROS) framework [17]. The specification includes:

- a kinematic description
- a dynamic description
- a visual representation
- a collision model

Kinematic description
The kinematic description consists of a list of link elements (links) and joint elements (joints) that connect the links. Joints can connect links rigidly or with a single degree of freedom where a single degree can be either fixed, revolute, prismatic or continuous.
Dynamic description
The dynamic description tells about inertial properties such as link mass and friction coefficients.
Visual representation
Although a visual representation usually is associated with Computer Aided Design (CAD) or 3D-modelling, in URDF it describes geometric properties like cylinder, box, dimension and colour.
Collision model
The collision model is used to detect collision with itself or/and an external object.

The vision-based control algorithm in this work uses a camera rigidly attached to the end effector, a construction also known as eye-in-hand. For this purpose, a camera is added to the the simulation. The most important functional properties of this camera are

- image resolution: 640×480 pixels
- RGB output format
- a frame rate of 30 frames per seconds which is the actual rate of the real camera

For this work, the location of the camera is on the gripper, which will not be used. Reason for locating it on the gripper is that for object detection an image is preferrably without occlusion.

The physical properties of the camera are also added to the model of the PERA. To this end, both a physical link and a fixed joint for the camera are added to the file that describes the arm. The joint defines the exact location of the cameralink on the hand. Located on the gripper, it is actually off the hand but still fixed to it. Result of this construction is that this way, the focal axis of the camera is aligned with the x axis of the hand. If the joint would connect the camera to a finger of the gripper, its focal axis would not be aligned as such and thus not point in the direction of the object.

The Robot Operating System is software developed by WillowGarage [18] and provides libraries and tools for building robot software. It uses nodes and topics or/and services for communication between those nodes.

A node is an application either provided by ROS or created by a developer. A topic is a message that can be published by and be subscribed to by nodes and usually provides floating point data. A service only acts when it is called, so a node providing a service will be inactive until then. It contains a clear text message and a request to and a response from a service node.

2.2 Kinematics of the PERA

2.2.1 Introduction

The kinematics of a robot describes the motion of a manipulator without considering the forces and torques that cause the motion. It can be divided into forward kinematics and inverse kinematics.

Forward kinematics is determining the position and orientation of the end effector with the values (angles) of the joints of a manipulator. Inverse kinematics can be divided into inverse *position* and inverse *velocity* kinematics.

Inverse position kinematics is the calculation of the joint angles from the position and orientation of the end effector and using these angles to control the arm using forward kinematics. Inverse velocity kinematics is the calculation of the joint velocities that in the end make the desired end effector velocity.

2.2.2 Inverse kinematics

In this thesis, inverse kinematics is used. The reason is that both the simulation and the real world are defined in cartesian coordinates. These are uniquely numerical coordinates in the form of $(x\ y\ z)$. This is easier to work with than joint angles of each of the seven joints and, moreover, people are used to working with them.

To control an end effector, the joints between the end effector and a chosen origin also have to be controlled. So with velocity control, the velocity of the end effector is directly related to the velocity of the joints between the robot torso and the end effector. The inverse joint velocity kinematics is calculated according to the equation from Siciliano et all [19] which is for a nonredundant nonsingular manipulator.

$$\dot{q} = J^{-1}(q) \begin{bmatrix} \dot{p}_d + K_p e_p \\ L^{-1}\left(L^T \omega_d + K_O e_O\right) \end{bmatrix} \tag{1}$$

In this equation

- $\dot{q}$ is the joint velocity
- J^{-1} is the inverse Jacobian
- $\dot{p}_d$ is the desired end-effector velocity
- K_p and K_O are positive definite matrices
- e_p the position error
- ω_d is the desired angular velocity
- e_O the orientation error. $e_O = \phi_d - \phi_e$ or the desired minus the computed set of Euler angles.
- L is the interaction matrix or image Jacobian matrix

This solution uses the geometric Jacobian and with that it avoids occurrences of representation singularities, which would happen with the analytical Jacobian. For calculating the position, the joint angle q is calculated using the generated trajectory, an initial position in radian, $\dot{q}$ and the time.

2.2.3 Transformations for the PERA

For kinematic modeling of the robot arm, coordinate frames must be established, so positions and orientations can be represented, as well as transformations among these frames.

A representation of this kinematic configuration can be obtained by using a homogeneous transformation which is a matrix representation of a rigid motion. For modeling a robot manipulator, the representation can be more simplified by using the Denavit - Hartenberg or DH convention. This convention assumes that

- a manipulator is made up of links and joints
- a joint is 1 DOF (Degree of Freedom) that is either revolute or prismatic
- there are n joints, starting from 1 to n where joint i connects link $i-1$ to link i
- there are $n+1$ links, starting from o to $n+1$ where o (origin) is the base and $n+1$ is the end effector
- link i moves when joint i is activated
- link 0 is fixed to the origin

As mentioned, the inverse kinematics is concerned with the finding of the joint variables in terms of the end effector's position and orientation. However, there may or may not be a solution that may or may not be unique. Eq. 2 taken from Sprong et all [20] shows a 4×4 homogenous transformation where R is a rotation matrix and o is an origin.

$$H = \begin{bmatrix} R & o \\ 0 & 1 \end{bmatrix} \in SE(3) \tag{2}$$

Then at least one solution has to be found for Eq. 3

$$T_n^0(q_1, \cdots, q_n) = H \tag{3}$$

where

$$T_n^0(q_1, \cdots, q_n) = A_1(q_1) \cdots A_n(q_n) \tag{4}$$

H is the desired position and orientation of the end effector. In Eq. 4, homogeneous transformation A_i is a product of four basic transformations which are shown in Eq. 5.

$$\begin{gathered} A_i = \mathrm{Rot}_{z,\theta_i}\mathrm{Trans}_{z,d_i}\mathrm{Trans}_{x,a_i}\mathrm{Rot}_{x,\alpha_i} = \\ \begin{bmatrix} cos\theta_i & -sin\theta_i & 0 & 0 \\ sin\theta_i & cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ \times \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\alpha_i & -sin\alpha_i & 0 \\ 0 & sin\alpha_i & cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \\ \begin{bmatrix} cos\theta_i & -sin\theta_i cos\alpha_i & sin\theta_i sin\alpha_i & a_i cos\theta_i \\ sin\theta_i & cos\theta_i cos\alpha_i & -cos\theta_i sin\alpha_i & a_i sin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{gathered} \tag{5}$$

In Eq. 5, θ_i, a_i, d_i and α_i are the parameters *joint angle*, *link length*, *link offset* and *link twist*, respectively for link i and joint i.

When a manipulator has at least six joints, it is possible to simplify the inverse kinematics problem by kinematic decoupling. This decouples the problem into inverse position kinematics and inverse orientation kinematics. Simplified for simulation, the PERA has seven joints, so the decoupling is applied and there are two sets of equations, representing rotational (Eq. 6) and positional (Eq. 7) equations:

$$R_7^0(q_1, \cdots, q_7) = R \tag{6}$$

$$o_7^0(q_1, \cdots, q_7) = o \tag{7}$$

The seven joints are revolute and therefore there are seven DOFs. This does not change with the addition of the joint between the hand and the camera because it is fixed. In reality, the shoulder, elbow and wrist are differential drives. For the seven joints in the simulation there have to be transformation matrices. These joints are

- shoulder roll
- shoulder pitch

- shoulder yaw
- elbow pitch
- elbow yaw
- wrist pitch
- wrist yaw

In Fig. 3, the simplified coordinate frames and in Fig. 4 the position extremes are shown.

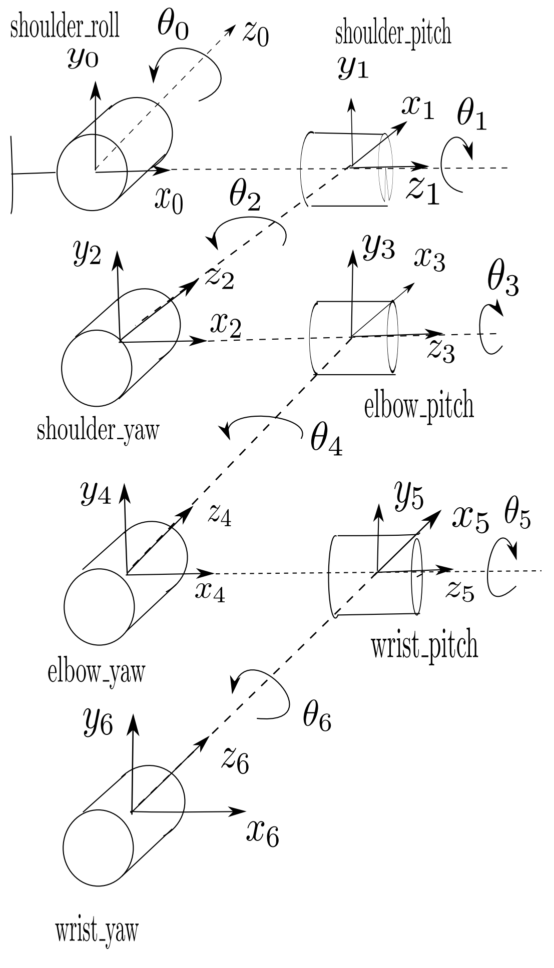


Figure 3: Seven coordinate frames of the PERA

The parameters according to the DH convention for the inverse kinematics of the PERA are shown in table 1. The link lengths a_1, a_2, a_3 are 0 because there is no distance between the joints and a_5 and a_7 are 0 because there is no distance to the respective pitch joints.
The joint limits for the PERA are shown in table 2.

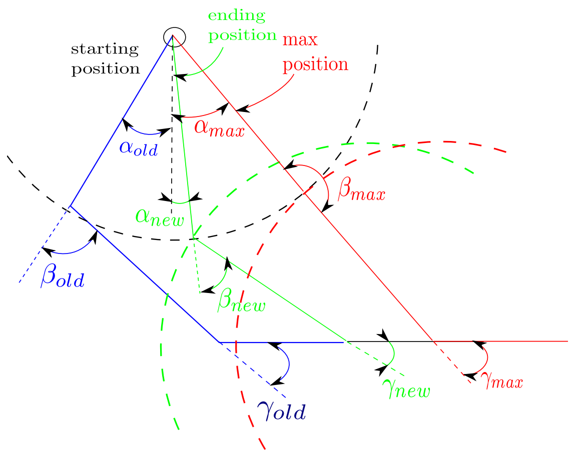


Figure 4: Position extremes of the PERA

Table 1: Inverse kinematics DH parameters for the PERA

Link	a_j	α_j	d_j	θ_j	Explanation
1	0	$-\pi/2$	0	θ_0^*	fixed to shoulder
2	0	$-\pi/2$	0	θ_1^*	pitch joint directly connected to roll joint
3	0	$\pi/2$	0.32	θ_2^*	yaw joint directly connected to pitch joint
4	0.32	$-\pi/2$	0	θ_3^*	
5	0	$\pi/2$	0.28	θ_4^*	yaw joint directly connected to pitch joint
6	0.28	$-\pi/2$	0	θ_5^*	
7	0	0	0	θ_6^*	yaw joint directly connected to pitch joint

a: length, α: twist, d: offset, θ: angle, * indicates variable

Table 2: Joint limits

Joint	$\theta_{min}[rad]$	$\theta_{max}[rad]$
θ_0	0	$\pi/2$
θ_1	$-\pi/2$	$\pi/2$
θ_2	$-\pi/2$	$\pi/2$
θ_3	$-\pi/2$	0.95
θ_4	-1.83	1.83
θ_5	-0.99	0.99
θ_6	$-\pi/4$	$\pi/4$

When axes $y1...y6$ are fully aligned, the PERA reaches the boundary of its workspace which correponds to a singular configuration or singularity. Identifying a singularity is important for a couple of reasons. For example, it represents a configuration from which a direction or motion may not be possible.

The problem is to find joint velocities $\dot{q}$ that give the desired end-effector velocity ξ with the Jacobian kinematic relationship,

$$\xi = J(q)\dot{q} \tag{8}$$

Note that J cannot be inverted in case of the PERA since this is only possible for a square Jacobian, which means a maximum of six joints. So in case of the PERA, the inverse velocity problem only has a solution if and only if $rank\ J = rank\ [J\ \xi]$. Since the number of simulated joints of the PERA is more than 6, it is kinematically redundant and the Jacobian cannot be inverted. There will only be a solution to Eq. 8 if ξ is in the range space of the Jaocbian. This implies that the inverse velocity problem can be solved using the pseudo-inverse of J. Now if $rank\ J = m$ and $m < n$, where m is the number of rows and n the number of columns, the right pseudo-inverse of J is

$$J^{\dagger} = J^T(JJ^T)^{-1} \tag{9}$$

and $JJ^{\dagger} = I_m$, where I is the unity matrix. Then the inverse velocity is calculated as

$$\dot{q} = J^{\dagger}\xi + (I_n - JJ^{\dagger})b \tag{10}$$

with $b \in \mathbb{R}^n$ being an arbitrary vector. J is computed using singular value decomposition:

$$J = U\Sigma V^T \tag{11}$$

where the columns of $U \in \mathbb{R}^{m\times m}$ are the eigenvectors of AA^T from a given matrix A and the columns of $V \in \mathbb{R}^{n\times n}$ are the eigenvectors of A^TA. Both U and V are orthogonal matrices and Σ is a matrix:

$$\Sigma \in \mathbb{R}^{m\times n} = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \sigma_m & 0 & \cdots & 0 \end{bmatrix} \tag{12}$$

In Eq. 12, $\sigma_i \geq 0$ are the singular values of the matrix and eigenvalues $\lambda_i = \sigma_i^2$. Then the right pseudo-inverse of J is shown in Eq. 13

$$J^{\dagger} = V\Sigma^{+}U^T \tag{13}$$

with

$$\Sigma^{+} = \begin{bmatrix} \sigma_1^{-1} & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2^{-1} & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \sigma_m^{-1} & 0 & \cdots & 0 \end{bmatrix} \tag{14}$$

2.2.4 Obstacle avoidance by Direct Trajectory Generation

The PERA has a reachable workspace, which is the space formed by all positions that the end effector can reach. In this space, objects may be present that have to be avoided. To avoid an obstacle, there are some methods to accomplish this:

- Using an online generated trajectory

- Using an offline generated trajectory
- Using controlled velocity
- Using a combination of methods

A trajectory is a path that has to be traversed within a certain amount of time. Online generation means that the trajectory is generated during the activity of the PERA. Advantage of this is that the trajectory can be changed if necessary but it requires a complicated algorithm. Offline generation means that the trajectory is generated prior to the activity of the PERA. Advantage is a simpler setup, disadvantage is that any disruption can cause unforeseen effects.

Trajectory generation is used to determine how the end effector has to move to avoid other objects on the way to the destination object. The use of Direct Trajectory Generation (DTG)[21] for obstacle avoidance means that constraints can be dealt with immediately. Among these contraints are the positions of the points along a trajectory, but also the maximum velocity at such a point. There are two algorithms for avoidance:

- point-to point or simple
- multi-point

With the point-to-point algorithm, there is a starting and end point, which is the avoidance point. For the multi-point algorithm, a via point is involved.

The DTG algorithms can be used in two ways: a vision determined starting point and a user given end point or the other way around. With the point-to-point algorithm, using a user given starting point and a vision determined end point, the velocity of the end effector has to be reduced to zero at a short distance from the object. So the control loop looks like Fig. 5, where q is the constraint vector. Elements q_f, $\dot{q}_f$ and $\ddot{q}_f$ form an avoidance motion.

$$q = [q_i, q_f, \dot{q}_i, \ddot{q}_i, \dot{q}_f, \ddot{q}_f] \tag{15}$$

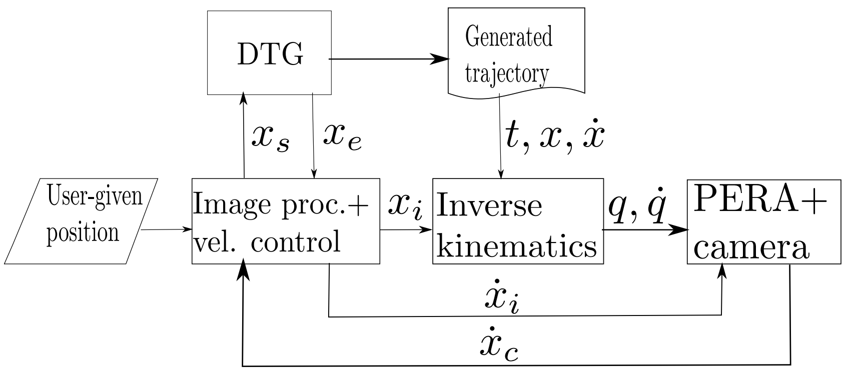


Figure 5: Control schema

This vector consists of

- initial point q_i
- final point q_f
- initial velocity $\dot{q}_f$
- initial acceleration $\ddot{q}_i$
- final velocity $\dot{q}_f$
- final acceleration $\ddot{q}_f$

In this thesis, the simple trajectory is used. So after detection of the object, the end effector moves from the starting point in a straight horizontal line along the x axis. So the y and z coordinates do not change. In Fig. 5:

- $\dot{x}_i$ is the velocity
- $\dot{x}_c$ the current velocity
- x_i is the cartesian coordinate
- t is the time
- x is the position
- $\dot{x}$ is the velocity read
- q is the joint angle
- $\dot{q}$ is the joint velocity

The DTG algorithm generates a 4 column table consisting of the trajectory: time, position, velocity and acceleration. From this table, only the time, the position and the velocity are used.
The motion of the PERA along the trajectory is shown in Fig. 6 where t_s is the start time and t_e the end time of the trajectory. It uses pseudo-inverse and a Vandermonde matrix which is a matrix with the condition that each row has to be a geometric sequence. Such an $m \times n$ matrix looks like Eq. 16.

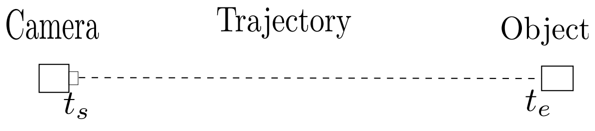


Figure 6: Trajectory sketch

$$M = \begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \cdots & \alpha_2^{n-1} \\ 1 & \alpha_3 & \alpha_3^2 & \cdots & \alpha_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_m & \alpha_m^2 & \cdots & \alpha_m^{n-1} \end{bmatrix} \tag{16}$$

Originally a stand-alone C program with the trajectory as output written in a text file, it is changed for this thesis into a program providing a ROS service that generates the trajectory and writes to file once called.

2.2.5 Determining the distance to an object

To avoid an object by using vision, it is important to generate a depth image. This depth is used to determine the distance to an object. The properties of coloured images contain the values x, y and z, which represent width, height, and colour. The depth of an image is not known, i.e. the depth of each pixel in the image.

The best way to generate a depth image[1], is using stereo vision, which generally utilizes two cameras. Stereo vision with one camera is possible as well, when images are taken from different positions, see Fig. 7.

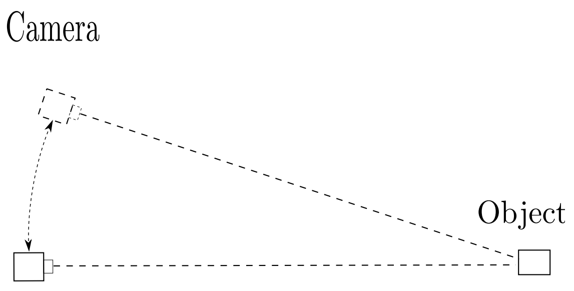


Figure 7: Determining the distance using one-camera stereo vision

When there is only one camera present at one position, the distance can also be determined. This is usually under the condition that the geometrical characteristics of the object are known. Downside of this approach is that the accuracy will be lower than with stereo vision. In this case there are the following options:

- image-based visual servoing
- prior calibration with the object
- using the perimeter of the object

Image-based visual servoing uses a vector of object feature parameters and uses e.g. the Speeded Up Robust Features (SURF) method, proposed by Bay et all [22]. This method consists of 2 steps:

- fixing a reproducable orientation based on information from a circular region around the interest point
- construct a square region aligned to the selected orientation and extract the SURF descriptor from it

With this option, the geometric charateristics do not need to be known in advance. Disadvantages are that end effector trajectories cannot easily be predicted and it may produce collisions with obstacles.

With prior calibration, the width in pixels of each object has to be measured at a known distance. By using the focal length of the camera, the next time the distance is known by determining the width in pixels again. Disadvantage of calibration is that it has to be repeated for each object.

[1]Siciliano, p. 409

When using the perimeter, the distance is determined by subsequently thresholding the image, applying the canny algorithm, find the contours (perimeter) and calculate its length. Then the ratio of the perimeter and the prior known geometric characteristics of the detected object is the distance. The larger the perimeter, and thus the smaller the ratio, the shorter the distance to the object is. Disadvantage of using the perimeter is that its calculation has varying results.

2.2.6 Velocity of the end effector

The control of the velocity of the end effector is important for two reasons. The first reason is that it enables the PERA to stop safely before reaching the object. The second reason is that the motors used for the PERA have a maximum safe speed. If this speed limit is broken, the motors will generate more power than the power supply. When this happens, this over-power might damage the motion controller boards. Table 3 shows the limits of the angular speed of the motors as stated in the manual.

Table 3: Maximum motor speeds in rad/s

	shoulder 1 & 2	shoulder 3	elbow 1 & 2	wrist 1 & 2
speed	1.47	2.27	2.8	3.54

The velocity of an end effector can be divided into two velocities:

- angular velocity
- linear velocity

The angular velocity is the velocity with which it rotates around its joint axis and is measured in either radians per second (rad/s) or revolutions per minute (rpm) where 1 rad/s = 0.104720 rpm. The linear velocity is the velocity along a straight line and is measured in m/s. For completeness, the relations between the velocities are given. For the used eye-in-hand system, the camera velocity is expressed as

$$\xi = \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{17}$$

where v is the linear velocity and ω is the angular velocity. The relation between the joint velocity and the velocities in Eq. 17 is the Jacobian J: $v = J_P(q)\dot{q}$ and $w = J_O(q)\dot{q}$ [23].

3 Architecture and Implementation

3.1 Architecture

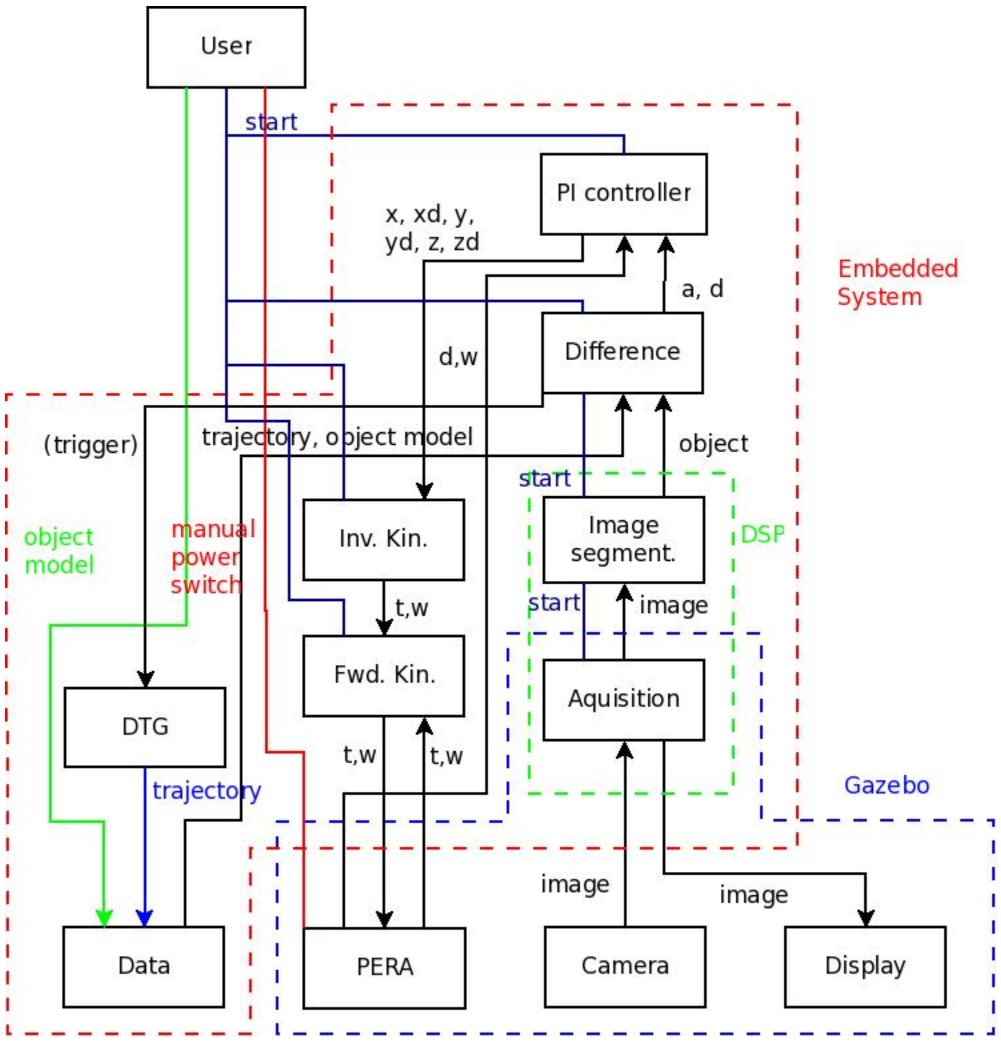


Figure 8: Global architecture

In Fig. 8 the global architecture is shown, where t = θ, w = ω, xd = $\dot{x}$, yd = $\dot{y}$, a = α, d = distance. The architecture has a commandline user interface, software and hardware modules, and can be simulated partially.

Next, the modules, functions and interfaces in the architecture are described from the bottom upwards. Unless stated otherwise, where the interfaces ”write”, ”read” and ”send” are used in combination with software modules, ROS messaging is used between the respective modules.

Description of module functions

• The function of the module ”Data” is to contain both the textfile holding the model of the object being detected and the textfile holding the trajectory generated by the DTG.
• The module ”PERA” hides the physical PERA as well as the simulated version.
• The module ”Camera” hides both the simulated and the physical camera.
• The module ”Display” hides a simulated as well as a physical display.
• The function of the module ”DTG” is to generate the trajectory along which the PERA will have to move.
• The function of the module ”Fwd. Kin.” is to read, write and compare the angles θ_i and velocities ω_i.
• The function of the module ”Aquisition” is to capture the image from the camera and send it both to the display and to the module ”Image segment.”. The module is started by ”Image segment.”.
• The function of the module ”Inv. Kin.” is to calculate θ_i and ω_i.
• The function of the module ”Image segment.” is to perform image segmentation by thresholding.
• The function of the module ”Difference” is to calculate the angles of the PERA and the distance of the Tool Center Point to the object.
• The function of the module ”PI controller” is to control position x_i, y_i, z_i and ω_i.
• The function of the module ”User” is to provide the commandline interface that allows all modules to be started and ended as well as enabling the user to fill the module ”Data” with object model data.

Description of the module interfaces

• *”Data” module*: the object model and trajectory are read by the module ”Difference”.
• *”PERA” module*: θ_i, ω_i and distance d(x_i, y_i, z_i) are read by modules ”Fwd.Kin.” and ”Difference”.
• *”Camera” module*: the image is captured using a ROS camera driver.
• *”Display” module*: it receives the image from the ”Aquisition” module.
• *”DTG” module*: after being triggered by the module ”Difference”, it writes the trajectory data to a textfile.
• *”Fwd.Kin.” module*: it reads the current angles θ_i and velocities ω_i from the modules ”PERA” and ”Inv. Kin.”. It writes angles θ_i and velocties ω_i to the PERA. The module is started from the commandline by the user.
• *”Inv.Kin.” module*: it reads the PERA coordinates and velocities from the controller, and afterwards it sends the results to ”Fwd.Kin.”. The module is started by the user from the commandline.
• *”Image segment” module*: via a variable it receives the captured image from the ”Aquisition” module, and sends the segmented image to the module ”Difference”.
• *”Difference” module*: It reads the trajectory and object model from the files contained in the module ”Data”. Via a variable it sends d(x_i,y_i,z_i) and the actual model size to the module ”PI controller”. After a positive trigger, the module ”DTG” is called. The module is started from the commandline by the user.

• *"PI controller*: it reads d(x_i,y_i,z_i) and ω_i from the module "PERA", and α_i and d(x_i,y_i,z_i) from the module "Difference". The module sends x, $\dot{x}$, y, $\dot{y}$, z and $\dot{z}$ to the module "Inv.Kin.". The module is started by the user.

Mapping of the modules

In the ideal non-simulation situation, the modules for image aquisition and image segmentation are on the DSP, which is inside the dashed green area.

When the Gazebo simulation is active, the modules for PERA, camera, display and image aquisition are performed by the simulation, which is inside the dashed blue area. The rest of the system is outside the simulation.

The software and data inside the dashed red area is on the embedded system. Outside the dashed areas, the user has to start the software, fill the datafile with the object model and either manually switch on the PERA powersupply or start the simulation.

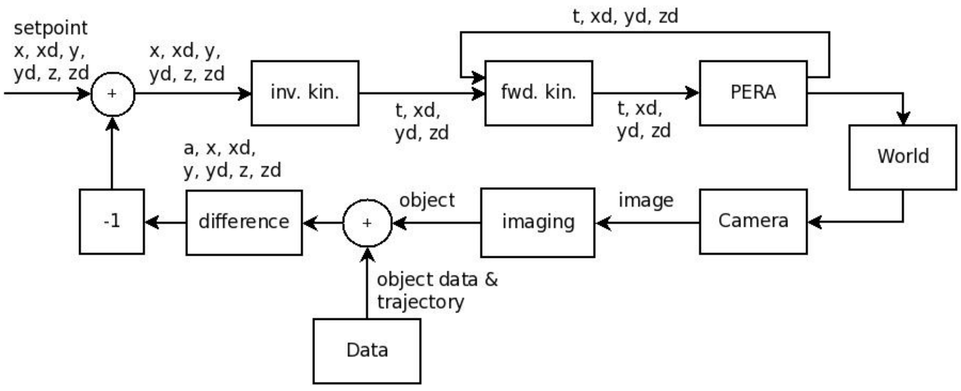


Figure 9: PI control schematics of the implementation

In Fig. 9 the control loop is shown. After the start, a first set of data is sent to *inv.kin.*. This sends angles θ_i and velocities ω_i to *fwd.kin.*. From here this data is sent and recieved in feedback. *PERA*, *World* and *camera* are necessary in this process to get the image that has to be processed by *imaging*. This contains both aquisition and processing. Based on the processed image, the object data and the trajectory, the angles θ_i and velocities ω_i are determined. The "-1" block is an integral action, which makes a controller error 0 if it reaches a steady state. According to its manual, the PERA has a PID controller which is part of the Xenomai real-time software and checks for error conditions on maximum velocity and acceleration. Since the PI controller checks on the actual values outside Xenomai, both controllers do not interfere with eachother.
Fig. 10 on page 24 shows the setup of the embedded system in the ideal situation when the DSP is used. In this situation the camera is directly connected to the DSP via the special camera connector. The embedded system's manual does not state its standard or protocol. Fig. 11 shows the actual setup of the hardware. As can be seen, both the PERA and the camera are connected to the embedded system the same way while the simulation runs on a laptop, which is

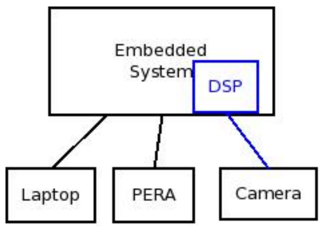


Figure 10: The ideal set-up for the embedded system

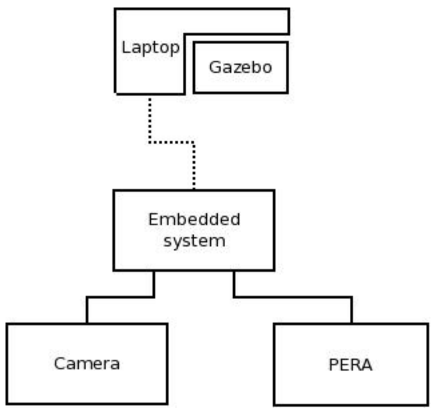


Figure 11: Block schema of hardware setup

an HP Elitebook 8530p with 4 GB RAM. Simulation is a way to find out if the combination of a model and an algorithm can do the desired task. After a successful simulation, experiments can prove whether the hardware and software are indeed capable of the tasks.

Boundary condition and convention

The experiments on the physical PERA and simulations on the virtual PERA

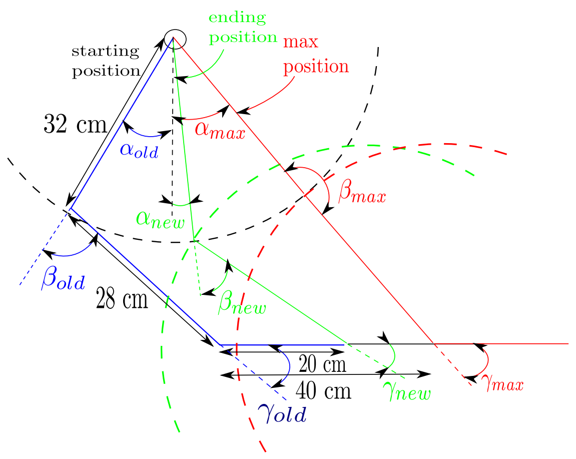


Figure 12: Position extremes of the PERA with lengths

are done under the condition that the detectable object is within the reachable workspace of the PERA. In contrast to the Amigo, the PERA is during the experiments mounted at a static position. Therefore, considering the lengths of the arm and end effector, shown in Fig. 12, any object further away than 40 cm from the end effector is out of the reachable workspace when keeping the end effector horizontally.

There are several conventions for roll, pitch and yaw. The convention used for this thesis is the same as stated in the manual of the PERA and as shown in Fig. 13 on page 25.

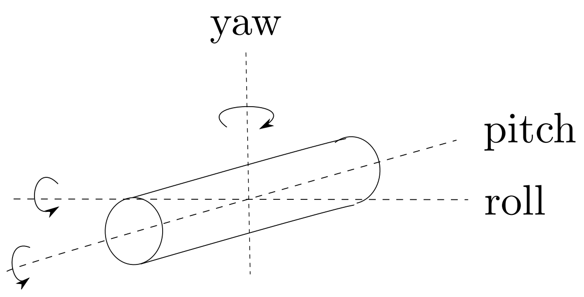


Figure 13: The followed convention for roll, pitch and yaw

The manual of the PERA recommends a setup where the PERA is the only device connected to the USB port. However, since this recommendation is from 2010, experiments have been done to see if this recommendation still holds.

3.2 Implementation

All modules are implemented in ROS nodes. A ROS node is a program that uses ROS to subscribe or/and advertise to ROS topics. A ROS topic is a message with which messages are sent between ROS nodes. Another possiblity is the use of a ROS service. In this case, an inactive ROS servicenode is called by another ROS node and after providing its service it will return to inactivity.

3.2.1 Forward kinematics

This node subscribes to the ROS topic advertised by the inverse kinematics node and publishes the joint angles to the PERA after which the end effector is placed in the position with the desired carthesian coordinates. These advertised coordinates are used as feedback by the image processing node.

Angles

`+callback(): void`
`+Angles(): constructor`
`+run(): void`
`+main(): int`

Figure 14: Class diagram for forward kinematics

In Fig.14 the Angles class is shown, which is in the forward kinematics node `arm_jref_gen`. The `callback` function receives the data from the inverse kinematics node and the PERA. The `Angles` constructor defines the subscription to ROS messages and the publishing of ROS messages to other ROS nodes. It also calls the callback function. The `run` function makes it possible to pusblish the data to the PERA.

3.2.2 Inverse kinematics

The algorithm used for the inverse kinematics is based on formula 3.89 for inverse velocity kinematics from Siciliano and calculates the joint velocity $\dot{q}_i = \omega_i \times J^{-1}$. Implemented in Matlab by dr. Dragan Kostić, it was ported to C++ for this thesis. Reason for porting instead of interfacing Matlab and C++ is that one can run only binaries autonomously on the embedded system. If Matlab exported C code is used, Matlab is needed as well and besides the consumption of extra resources, there is no Matlab version for the ARM processor. When Matlab is used externally, it only supports the use of embedded system I/O by Simulink.

In Matlab all variables are either vectors or matrices, and any conversion between them is done automatically. For C++, the Eigen3 library is designed to use linear algebra. It enables the use of vectors and matrices with either fixed or variable size. The class diagram in Fig. 15 shows the structure.

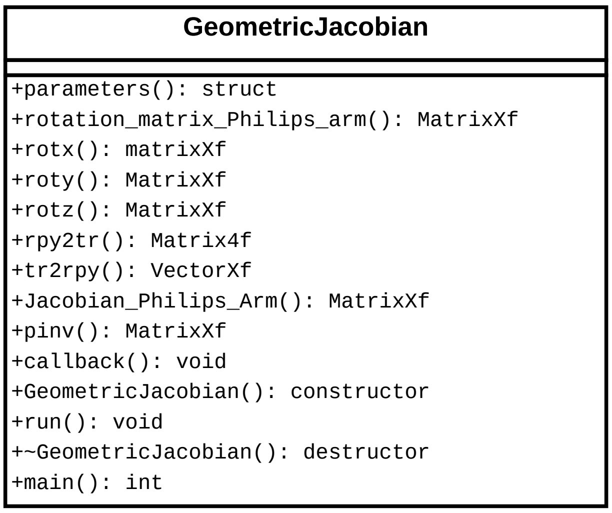


Figure 15: Class diagram for inverse kinematics

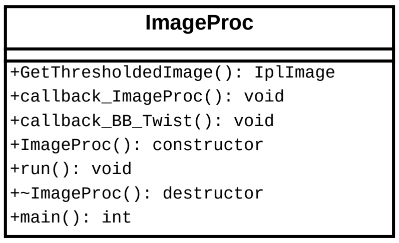
The `rotation_matrix_Philips_arm` function calculates a rotation matrix for the PERA and is called by the `callback` function. The functions `rotx`, `roty` and `rotz` return a homogeneous transformation representing a rotation of θ_i about their respective axis. The `rpy2tr` function returns an homogeneous transformation for the specified roll/pitch/yaw angles. These correspond to rotation about the Z-Y-X axes respectively. This function calls for the rotx/roty/rotz functions. The `tr2rpy` function returns a vector of roll/pitch/yaw angles corresponding to the rotational part of the homogeneous transformation TR. The `pinv` function is a C++ solution for the Matlab `pinv` function. It is taken and adapted from the trajectory generation code. All these functions are used or called by `callback`. The struct `parameters` defines the dimensions of the arm, centers and weights of mass, mass moments of inertia and gravitational acceleration.

The node reads the x coordinate and the velocity $\dot{x}$ at that point from the textfile generated by the DTG node, calculates both the joint velocity and angle and sends them to the forward kinematics node by advertising the rostopic `joints_angle`.

3.2.3 Image segmentation

This node is responsible for capturing the camera image and determining the distance d(x, y, z) of the end effector to the object. For the communication with the inverse kinematics node, the ROS topic `im_proc` was created.The structure of this node is shown in Fig. 16. The following functions are performed:

- image segmentation
- determination of the image perimeter
- determination of the object perimeter
- detection of object keypoints
- calculation of the geometric characteristics of the object by using contents from the user provided `geochat.dat` file.



```
+GetThresholdedImage(): IplImage
+callback_ImageProc(): void
+callback_BB_Twist(): void
+ImageProc(): constructor
+run(): void
+~ImageProc(): destructor
+main(): int
```

Figure 16: Class diagram for image processing

The objective of image segmentation is to make it easier to analyse a digital image by partitioning it into objects. The analysis is done by assigning a label to objects or boundaries with the same characteristics. After this analysis, all labels that are not wanted are removed. Next, the remaining object in the image can be processed further to determine its size.

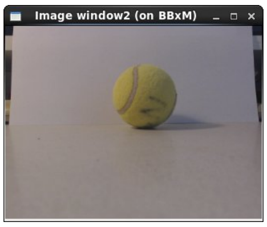


Figure 17: Image of tennisball by daylight

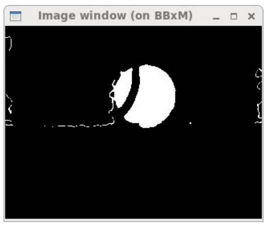


Figure 18: Segmented image of same tennisball by daylight

On page 27, Fig. 17 shows a normal image of a yellow tennisball and Fig. 18 shows the same tennisball after segmentation.

The segmentation is done by thresholding, which means that the colour value of 1 colour is used to separate everything with that colour from the image. The resulting binary image shows the separated object in a black image. After the segmentation, two methods can be followed to calculate the distance to the object. For a successful calculation, the condition is that the distance between the end effector and the detectable object is no further that 40 cm. Reason for this is the reachable workspace limit as can be seen in Fig.12 on page 24. One method is to determine the number of pixels of the perimeter. This is done using the `findContours` and `arcLength` functions.

Using the length of the contour, the decreasing ratio of the full contour and the object properties is a measure for the distance. Generally, the determination is influenced by the position of the camera with respect to the end effector. The end effector consists of the hand and a gripper. The length of the gripper is 10 cm, so if the camera is placed on the hand, the captured frame will correspond to the coordinates of the hand.

If the determined object location is further than it actually is, the gripper could make contact with the object while avoidance is the objective. So apart from the reason mentioned in the chapter on kinematics, this is another reason why the camera will be positioned on the gripper.

A second method is keypoint detection. The total length of distances between the keypoints of the perimeter is a measurement of the perimeter length in pixels. From this, and the ratio of the complete image, the distance is measured. An example of keypoints is shown in Fig. 19. In this image, the small circles are the keypoints.

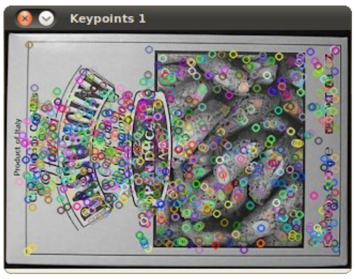


Figure 19: Example of keypoints taken from [24]

For the implementation of this node, OpenCV is used. OpenCV is a library that contains an API for different programming languages. API is the abbreviation of Application Programming Interface and defines functions needed to get a specific task done. In this thesis C++ is used and although C++ is a subset of C, the API's are different in detail.

In Fig.16 on page 27, `GetThresholdedImage` performs the image segmentation. The `callback_ImageProc` function receives the image, calls the image segmentation function and does the distance calculation. The `callback_BB_Twist` function receives the calculated distance and controls the velocities. The `run` function sends the data using ROS messaging.

Since the adjustment of the velocity depends on the distance determined by the image processing, this takes place in this node as well. The camera velocity, shown in Eq. 18, contains both the linear and the angular velocity.

$$\xi = \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{18}$$

With an eye-in-hand system, the camera can be attached either above or below the wrist. In the latter case, the camera only observes the object and this approach is chosen for this thesis.

Although eye-in-hand systems in general have the origin of the camera frame move with linear velocity v and the camera frame rotate about the ω axis, , the DTG algorithm only uses translation. This means that the camera is positioned right in front of the detected objected, after which the distance is determined.

3.2.4 Trajectory generation

Cartesian coordinates contain position (x, y and z coordinates) as well as orientation (roll, pitch and yaw). As mentioned earlier, the trajectory works by translating along one cartesian coordinate. Reason is that the DTG works in a 2D image plane in which there are only x and y coordinates. So when moving along more than one axis, a separate calculation is needed for each axis.

Unlike the other nodes, this node provides a ROS service and is called from the image processing node until the call is successful and the trajectory is generated. The trajectory is written into a newly created file on disk which is read by the inverse kinematics node. In Fig. 20 the class diagram is shown.

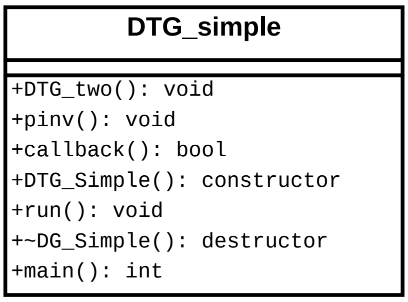


Figure 20: Class diagram of the simple version of DTG

The dashed lines in Fig. 21 indicate that either one or the other happens, not both at the same time.

Listing 1 shows that as long as the *boolean* variable `traject` is false, the service will be called until `traject` is set true. The request of the service is the starting x coordinate of the gripper, the responses are the x coordinate of the end point and the velocity of the end effector at the end point.

The output of the trajectory generation in 1 direction is a table consisting of 4 columns as shown in Table 4.

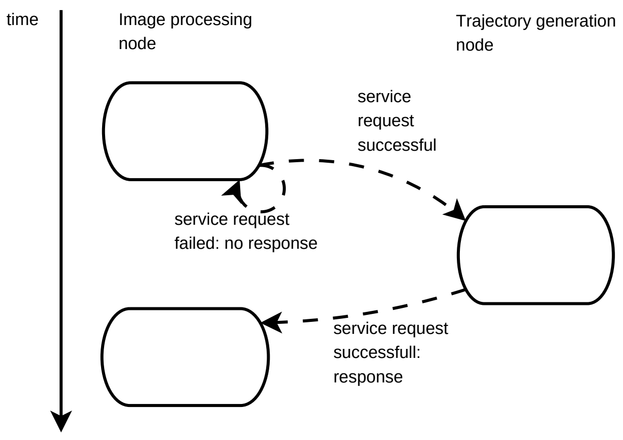

time
Image processing node
Trajectory generation node
service request successful
service request failed: no response
service request successfull: response

Figure 21: Schematic of a ROS service request

Listing 1: Code snippet of the call for trajectory generation

```
boolean no_traject = true;

...
while (no_traject)
{
    if (dtg_client.call(srv))
    {
        ...
        no_traject = false;
    }
    else
    {
        ROS_ERROR("Call to DTG failed");
    }
    ...
}
```

Table 4: Columns of the original DTG trajectory

time	x	$\dot{x}$	$\ddot{x}$
1.000	...	...	...
⋮	...	...	...
0	...	...	...

4 Embedded System

4.1 Introduction

Embedded systems are not described by a single definition. According to Vahid et all [25], one can state that any computer other than a desktop computer, is an embedded system. Strictly speaking, this means that laptops should be regarded as embedded systems although they are not accepted as such. Another description can be that an embedded system is a computer that controls a device that is not a computer. Examples for this one are a microwave oven, a washing machine or a board computer. A property that goes for most embedded systems is that they are usually based on ARM or (Atmel) AVR[2] processors.

4.2 The board

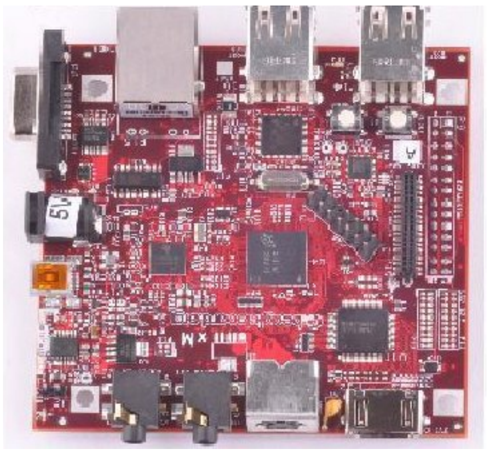


Figure 22: Image of the BeagleBoard-xM taken from the referencemanual

The BeagleBoard-xM (BBxM) shown in Fig. 22 is a ARM-based board with a clock-frequency of 1 GHz. It has a Texas Instruments Cortex-A8 compatible Open Multimedia Applications Platform (OMAP) DM3730CBP System-on-Chip (SoC) processor. The Cortex-A8[3] is a dual-issue superscalar CPU which means that it is single-core using instruction level parallellism [27]. It is also a RISC CPU, which means that it uses a reduced instruction set, compared to for example Intel CPUs. Also, the Cortex-A8 is single-threaded. This means that only one instruction can be executed at a time.

The OMAP3 is a heterogeneous multicore CPU because it is divided into the ARM general purpose processor (GPP) and the TMS320C64x (C64x+) Digital Signal Processor (DSP), with an L2 cache up to 1 MB, and 512 MB Low-Power Double Data Rate (LPDDR)[4] Random Access Memory (RAM)[5].

The BBxM is targeted at the Open-Source (developers) community and not intended for use in end-products. As such it is an board suited well to use in experiments. Apart from the mentioned DSP, the BBxM has the following properties:

[2]There is no official explanation of what AVR stands for, but it is commonly accepted that it stands for **A**lf (Egil Bogen) and **V**egard (Wollan)'s **R**ISC processor, where Alf and Vegard are the original architecture developers.[26]

[3]A indicates Application processor

[4]Because of its low-power it is very suitable for mobile use

[5]the volatile working memory of a computer

- RS232 serial connection
- SMSC 9514 Combined USB/Ethernet
- DVI-D (digital video) output connection
- 512 MB RAM

Electrical demands show that the BBxM is very energy friendly, which complies with a low cost solution. However, using the USB A ports requires that:

- the power input must be 4.75V-5.25V (ideally 5.0V).
- the USB hub must have a power cable and can not be powered by a USB cable.
- 400mA for the BBxM plus up to 500mA per USB device. USB cameras use around 150mA.

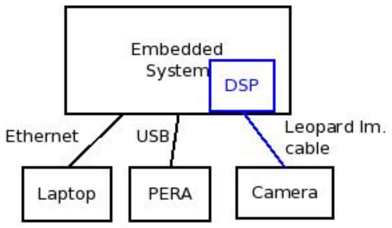


Figure 23: Ideal set-up for the embedded system with connectiontypes

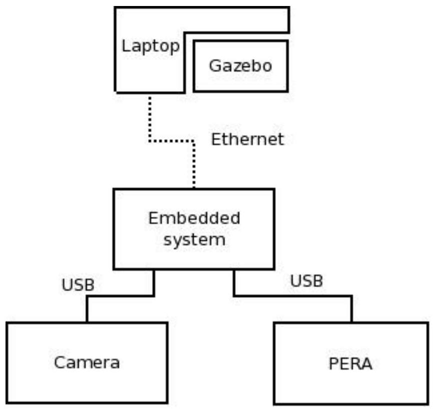


Figure 24: Actual schema of hardware setup with connection-types

The place of the BBxM in the hardware setup is shown in Figs. 10 and 11 on page 24. However, because of the difference between the ideal setup and the actual setup, the connections are shown in Figs. 23 and 24. The Leopard Im. cable in Fig.23 fits to the earlier mentioned 40 pins connector and delivers the images directly to the DSP. The actual setup can cause connection problems because the ethernet connection is an ethernet simulation over USB which is dealt with by the earlier mentioned SMSC 9514 chip.

4.3 Embedded Linux

As with all embedded systems, the use of an operating system is required. The choice is basically the same as for standard computers so one can choose Windows Embedded or Embedded Linux. Because of its open character, Embedded Linux is chosen. Linux is offered in many so-called distributions, of which three were considered because of their use by the community for the BBxM:

- Android [28]
- Ubuntu Linux [29]
- Ångström Linux [30]

According to Perneel et all [31], Android is not qualified for use in real-time environments. Therefore the choice is between Ubuntu and Ångström. Ubuntu fully supports ROS and, depending on tools, kernel-version and -configuration, support for DSPLink while Ångström has full support for DSPLink and experimental support for ROS. Because both the DSP and ROS have a keyrole in this thesis, experiments are done with both Ubuntu and Ångström.

4.4 Real-time

The PERA works real-time, so the software using the PERA has to be able to deal with real-time as well. Real-time can be divided into hard real-time and soft real-time.

In the case of hard real-time, when an event can occur in e.g. a situation of life and death, there is a deterministic deadline. In case of soft real-time, there is a generally met deadline [32].

The Linux kernel is not real-time. Real-time functionality on Linux can be achieved by adding real-time software to the kernel. The real-time software used by the PERA driver is Xenomai [33] which is "...a real-time development framework cooperating with the Linux kernel, in order to provide a pervasive, interface-agnostic, hard real-time support to user-space [6] applications, seamlessly integrated into the GNU/Linux environment."

To be able to run the Xenomai sub-kernel with the Linux kernel, three Xenomai packages have to be installed and the Linux kernel source has to be patched with Xenomai prior to building it. These packages are:

- linux-patch-xenomai, the kernel patch that integrates Xenomai with the Linux kernel
- libxenomai1, the driver that has to be loaded to access Xenomai
- xenomai-runtime

Since a computer can normally cope with only one kernel, there has to be a way to control the cooperation between the computerhardware, the Linux kernel and the Xenomai sub-kernel. This control is often taken care of by a hypervisor, which is a software, firmware or hardware that creates and runs virtual machines.

The hypervisor function is taken care of by Adeos, which is a nanokernel HAL (hardware abstraction layer) that operates between computer hardware and the operating system that runs on it. Figure 25 shows how Xenomai is integrated with the Linux kernel. ADEOS is the abbreviation of "Adaptive Domain Environment for Operating Systems", ipipe stands for "interrupt pipeline".

[6]User-space is "the memory area where all user mode applications work and this memory can be swapped out when necessary."

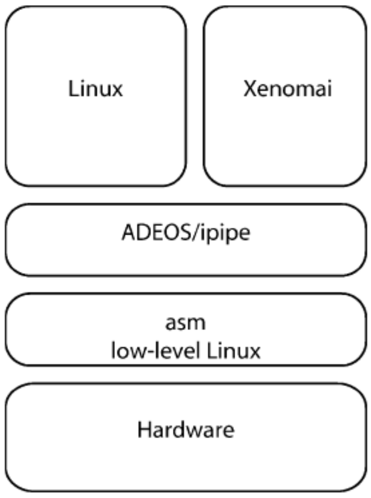


Figure 25: Graphical explanation of Xenomai/ADEOS

4.5 Computer vision

As the velocity control in this thesis is based on vision, a real-time and efficient computer vision application progamming interface (API) is necessary. Such an API is available as OpenCV [34] which is "designed for computational efficiency and with a strong focus on real-time applications".

Since OpenCV is available for ROS, it is easy to use in nodes. It can also efficiently use the memory, which is very important for an embedded system. For example, the OpenCV `Mat` class automatically allocates output image memory allocation for OpenCV functions. In this thesis the API is used to capture and process images in aid to determine the distance to an object.

4.6 Camera

As mentioned earlier, this thesis uses a trajectory generation algorithm developed by Roel Pieters [12]. The equipment used for his work is shortly reviewed here.

The camera that is used is the Prosilica GE680M, targeted at the professional market and unavailable during this thesis. The available models are GE680 and GE680C which both have a resolution of 640×480. The only difference between these cameras is that the standard model delivers monochrome images whereas the "C" model delivers colour. Another important property is the framerate of 205 fps.

The BBxM has a 40 pins connector that supports camera modules from Leopard Imaging [35], for example the LI-5M03. This way, a camera is connected directly to the DSP, which makes image processing much faster than when a USB camera is used. The modules are mounted directly on the BBxM. However, the driver for the LI-5M03 requires an older kernel, while ROS requires a newer kernel on which a USB camera can work. As a consequence, it is chosen to use a USB camera. Because of the proprietary driver, the protocol used between the DSP and the camera is unknown.

The camera used for this thesis is a Logitech HD Pro Webcam C920, see Figs. 26 and 27. Its maximum resolution is 1920×1080 with a maximum framerate of 30 fps. Because of this framerate, the camera simulation in Gazebo is set at 30 fps. This setting can give a problem with the quality of the captured images. Reason is that with a mains frequency of 50 Hz, an image captured at 30 fps will not always have a constant result. In simulation, this problem does not occur.

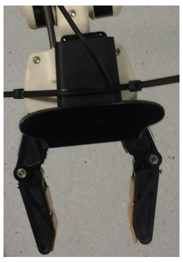


Figure 26: Logitech Pro Webcam C920: topview

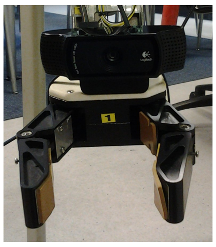


Figure 27: Logitech Pro Webcam C920: frontview

In order to get a good view of what the BBxM is grabbing and deciding on, it is best to view the camera images on a monitor, to test the properties of the camera prior to using the ROS driver, `Cheese` and `GUVCView` are used.

4.7 Image Processing

To be able to use the full capacity for image processing on the BBxM, the DSP should be used. Without using it, the BBxM will run the code but the images from the camera will be processed using the same memory and CPU as is used for ordinary data.

Like with any other hardware, a driver is necessary to get access to the DSP. There are three drivers capable for this:

- DSPBridge
- DSPLink
- SYSLINK

The choice is made by looking at the support for the SoC CPU. DSPBridge does not support the SoC DSP/GPP combination of the DM3730 while DSPLink does. SYSLINK supports access to the DSP from OMAP4 onwards, which makes DSPLink the driver to use.

In combination with DSPLink, DSP/BIOS LINK [36] provides a small firmware real-time library and tools for real-time analysis, runs on the DSP side and provides services such as:

- PROC (Basic processor control)
- Inter-Processor Communication protocols for different types of data transfer between the processors
- Inter-Processor Communication building blocks which are low-level building blocks used by the protocols

Communication between the GPP and the DSP works as follows:

- messages are send to the DSP and received back using DSP/BIOS LINK.
- the message contents are verified against the data sent to the DSP

DSPLink is an API from Texas Instruments and comes with tools for communication between the GPP and the DSP. Its latest release brings Linux kernel 2.6.37 with it. To get the DSP working, both DSP/BIOS and DSPLink are necessary.

The main processes are:

- a Linux application that loads and starts the DSP using PROC calls and exchanges messages and data through message queues (MSGQ) through a shared memory region which is configured using POOL.
- a DSP/BIOS application that starts after being loaded by the Linux application and a task (TSK) receives and sends messages back to it using the MSGQ and POOL structures.

4.8 Connectivity

Like other computers, the BBxM has several interfaces:

- a USB OTG port
- an ethernet connector
- an RS232 serial connector
- a USB-A hub

The USB OTG port is used to connect the BBxM to another computer by ethernet over USB. This can be another BBxM or any other.

By using the ethernet connector, it can be connected to any ethernet network. When inter-computer communication is needed, using the secure socket layer (ssl) is recommended. Commands using ssl are:

- secure shell (ssh): enables operating the BBxM over a secure connection from anywhere the connection to the BBxM is established.
- secure copy (scp): enables to copy files from and to the BBxM over a ssh connection.

With the serial connector, a terminal-like connection can be made with for example a laptop, depending on the Linux kernel. When there is no other connection available, and only textual output is sufficient, the BBxM can be managed as well using `minicom` on Linux and `PuTTY` on Windows. ROS messages can be sent over the serial port using `ros-serial`. However, for this thesis, this is not recommended to use. Reason is that on the laptop, ROS Electric is installed, while on the BBxM, ROS Groovy is installed and the ROS messages of different ROS versions are binary different so the messages are mostly likely misunderstood. ROS Electric is on the laptop because of the used version of the Gazebo/Amigo simulation, and it was not available for the BBxM where ROS Groovy had to be used.

The USB-A hub is suitable to connect various peripherals such as the PERA and a webcam. Experiments have to show if they can be connected simultaneously or not.

5 Simulations

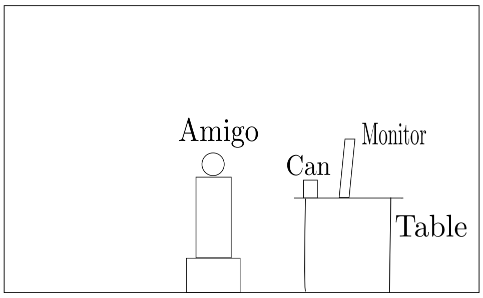


Figure 28: Sideview sketch of the Amigo simulation

As mentioned earlier, the simulations are done using Gazebo in combination with the Gazebo simulation plugin for ROS. Gazebo is built on top of the Open Dynamics Engine which is an open-source library. It allows the simulation of rigid body structures that are connected via joints. The graphical rendering is done by the Object-Oriented Graphics Rendering Engine. This 3D engine is open-source as well.

As mentioned earlier, the simulation is run on a laptop. It uses an environment that consists of the simulated Amigo with a can and a monitor on a table as shown in Fig. 28. For the simulation in this thesis, the monitor is not used. This environment is designed by the Robocup@home team of the TUe and for this thesis changed by

- relocating the can to coordinates $\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0.85 \\ -2.4 \\ 0.8 \end{pmatrix}$
- rotating the Amigo by 90° to the right, pointing the right-side arm in the direction of the can at the same y coordinate as the can.
- lifting the right arm of the Amigo in such a way that the end effector is at the same z coordinate as the can.

This way the starting position $\begin{pmatrix} 0.718 \\ -3.049 \\ 0.791 \end{pmatrix}$ of the end-effector should have a distance of 40 cm from the object. With the maximum reachable distance of the end-effector, this is a first test.

As described in the chapter on Robot Modelling, the URDF can contain collision information. During some simulations it was noticed that the PERA is able to go straight through the monitor as if it is not there. See the sketch in Fig. 28. But since the simulation will only include the can and not the monitor as an object that has to be avoided, this is of no concern.

In the original experiments for the DTG (see [37]), an object had been placed between the initial position and the desired end-position. The object avoidance causes the end-effector to reach a totally different end-position. This is not the case for this thesis since the desired end-position will be at close distance in front of the object.

5.1 RAM memory use

The amount of RAM in a computer has a major influence on simulations. The first basic simulations are run on 3 GB RAM and exists of running

- Gazebo
- the Amigo simulation
- the forward kinematics node arm_jref_gen
- the three camera nodes image_view_camera_out, image_view_camera_out_-seg and image_view_camera_out_perimeter
- command `rostopic echo /gazebo/link_states/pose[43]/position`, which displays the x, y and z carthesian coordinates
- Eclipse SDK 4.2.2 with CDT plugin (for C++ development)
- Java VM 1.7.0_25

The simulation runs from the start of the simulation at ROS simulationtime 5.339 till 12.54. The amount of RAM in use during this simulation is (round) 2.9 GB. In order the find the memory used by Eclipse, Eclipse is ended and thus removed from RAM. This shows that the difference in RAM use is only 2kB. Running the inverse kinematics node `pera_7dof_inv_kin`, the *top* command shows that it consumes 5560 kB. For extensive simulations, a larger amount of RAM is better.

5.2 Influence of the Linux kernel

The speed of any application on a computer depends on both CPU speed and the available amount of RAM. On 32-bits computers, a critical role is played by the kernel for addressing RAM. A standard 32-bits kernel is able to address a maximum of $2^{32} = 4294967296$ byte $\approx$ 4 GB. A 32-bits kernel with Physical Address Extension (PAE) can surpass that limit. On a 64-bits computer this is no issue.

On the laptop, the amount of addressed RAM is 3 GB while its BIOS shows that 4 GB is present. A check on the kernel shows that the installed kernel is a version without PAE. After installing a PAE-kernel, the *free* command shows 4 GB RAM. Running the same basic simulation as in the previous paragraph, the break-down of the memory use is shown in Table 5 on page 40.
Table 5 shows a very different use of RAM than with the non-PAE kernel. The total amount of RAM in use (1.07 GB) clearly does not even come close to 3 GB. It is not clear though, if this difference is completely explained by the kernel change. This is not further investigated.

5.3 Start and end of simulation

After starting the simulation environment, the actual movement in the simulation is started by the forward kinematics node. It is noticed that there if often a delay of multiple seconds before the PERA actually starts its movement. Since

Table 5: RAM use during basic simulation running on PAE kernel

Software	Used RAM (bytes)
Linux + 1 terminal	657137664
2nd terminal added	659906560
3rd and 4th terminal added	666406912
Amigo simulation added	994598912
arm_jref node added	1009180672
arm_im_proc node added	1045262336
5th terminal added and running echo_position	1073025024

there is no delay in the nodes, the assumption is that it is caused by either the Amgio simulation or by Gazebo.

The end of a movement in the simulation is often very slowly. This shows that there is a velocity control active in either the Amigo simulation or Gazebo. Since velocity control is part of the subject of this thesis, this built-in velocity control has to be disabled in order to be able to test its correct working without experiments on the real PERA. After examining the simulation software and scripts, the built-in velocity control could successfully be disabled by changing the start-script.

5.4 Memory allocation

All software that is used, is loaded into RAM first. Sometimes and at random moments, but after running the simulation for some time, the `image_proc` node aborts giving the error `terminate called after throwing an instance of ´std::bad_alloc´`. This error is no segmentation fault and thrown when the program is out of memory or due to a stack overflow. A stack is a memory area where data is added or removed according to the Last In, First Out (LIFO) principle. Being out of memory is not the case, since the other nodes keep running. Increasing the stack size by changing a software setting is a possible solution but it does not explain why only this node aborts, and the others keep running. Since the error is of intermittent nature, it is hard to determine the cause and it is not further investigated during this thesis.

5.5 Minimum and maximum angles

Both the Amigo simulation and the PERA have minimum and maximum angles for the joints. The minimum joint angle is when a joint is at its start position, the maximum joint angle is when a joint is set to its utmost position.

When applying the minimum and maximum angles of the PERA to the software, it shows that the roll of the shoulder is not responding correctly. It turned out that the direction signs in the Amigo simulation are reversed for the right-side PERA. So, as shown in Table 6, the range of the roll of the shoulder R_s as stated in the manual is different than in the simulation.
Presumably, the ranges in the manual are correct for the left-side PERA. Also, the values for the pitch movement of the elbow is very different in simulation as Table 7 shows.

Table 6: Difference in range between PERA and simulation. Angles in radians.

	PERA	simulation
min	0	-1.57
max	1.57	0

Table 7: Difference in angles (in radians) between PERA and simulation

	PERA	Simulation
min.	-1.57	0.01
max.	0.95	2.22

5.6 Erratic behaviour

It is commonly known that most software contains errors of some kind, even though it compiles without warnings or errors. Software that usually runs just fine, can show erratic results that can possibly cause dangerous situations. Software written for this thesis makes no exception.

The inverse kinematics algorithm mostly runs fine, with a start position at q_0 = [0, -0.45, 0, 1.6, 0, 0.4. 0]. However, sometimes it shows behaviour which is not immediately reproducible. The following situations occur:

- after reaching its start position, the PERA starts moving and does not stop.
- all angles become 0, resulting in the PERA stretched downwards. For the real PERA this is a different position, as decribed earlier.
- the PERA takes an entirely different position with very different joint angles and holds it position. See Fig. 29 for an example.
- after starting the `pera_7dof_inv_kin` node, instead of assuming its starting position q_0, one or more of the preset joint angles is set to 0. Despite restart, this can randomly happen once or multiple times after eachother.

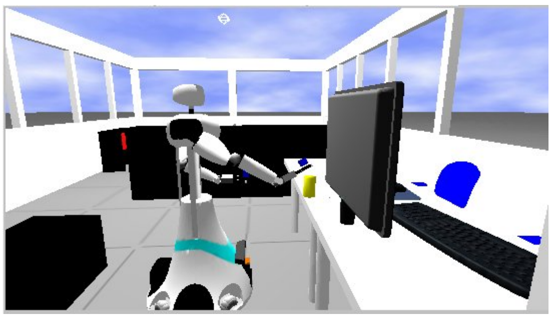


Figure 29: Erratic starting position

5.7 Position of the shoulder

The Amigo version used in this thesis dates from end 2012/start 2013. In the simulation, the height of the shoulder is flexible. If the arm is moving upwards and the end effector gets stuck under a rigid object, the upper body of the Amigo will slide into the lower torso. During the experiments, the PERA is mounted rigidly on top of a table and is not compliant to any displacing force. As a result, certain events during the simulation can not be compared completely with the real PERA.

5.8 Determination of the size

As described earlier, two methods to determine the size of an object are to use the length of its contour or to calculate the distances by using the keypoint locations. With hindsight, these methods are also part of blob object detection which could have been used. Examples such as [49] show it is possible.

The problem with using the first method is that neither the contour nor the arclength prove to be constant. To get the best approximation of its length, the contour and arclength are determined several times and the largest value is used.

The problem with the second method is that often, keypoints are not aligned in a single line along the perimeter but grouped together and thus can not be used to calculate the perimeter length of the object. Corner keypoint detection of the object is not possible either because those indicate corners of areas with equal values instead of corners of an object.

As such, determining both the size of and the distance to an object with one camera from one position is only feasible by approximation.

5.9 Direct Trajectory Generation

As stated earlier, the condition for a successful call to the DTG is such that the distance between the end effector and the detectable object is no further than 40 cm. In addition to that, the generated trajectory length is emperically adapted from 1000 to 285. Reason for this adaptation is that the x value of the trajectory is around 0.4 then, which is 40 cm.

Table 8: Results of trajectory generation with Direct Trajectory Generation

time	x	$\dot{x}$	$\ddot{x}$
1	2.5342e-08	7.59881e-05	0.151824
$\vdots$	$\vdots$	$\vdots$	$\vdots$
0.718	0.358696	3.13491	13.4103
0.717	0.361837	3.14831	13.377
0.716	0.364992	3.16167	13.3433

Table 4 on page 30 shows the generic contents of the output. Table 8 shows the results of the Direct Trajectory Generation algorithm according to Roel Pieters which works in 1 dimension.

The table shows that the velocity and acceleration increase until halfway, then decrease till 0. Since the end effector has to stop before the object is reached, the trajectory has to be generated with the end point located before the object as the sketch in Fig. 30 shows where A is the camera and B is the object.

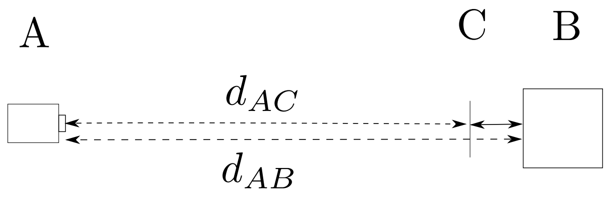


Figure 30: Trajectory between camera and object with avoidance location

After comparing it to the algorithm provided by D. Kostić, the length of the trajectory is changed back to its original 1000. Reason is that the x coordinate is calculated after measuring the distance so each coordinate will fit in.

The algorithm differs from the one provided by D. Kostić by the number of elements in the trajectory vector. DTG returns 4 elements per direction (x, $\dot{x}$ and $\ddot{x}$) and thus for a 2D trajectory has to be run twice. Since the inverse kinematics algorithm in this thesis uses both x and y coordinates, the vector returned by DTG is changed accordingly by adding y and $\dot{y}$ and setting them to 0.

5.10 Test trajectories

During previous tests, the PERA shows not to move despite of the generation of a trajectory. Therefore a test is done to see if the end effector (and thus the PERA) is able anyway to make a move following a given trajectory. First a test is done using the original algorithm and simulation in Matlab. Its track is replaced by 2 different trajectories and the result with both is that the PERA follows its newly given trajectory perfectly and in time. Next, a trajectory similar to the one created by the DTG is tested with good results.

Then the DTG-created trajectory is replaced by the same test trajectories in the Gazebo environment. Result is that nothing happens. The correct ROS messages are sent with the correct values, but there is no movement. So while all tested trajectories are clearly good, some mechanism in the Gazebo simulation prevents the trajectories from being executed. I was unable to find out which mechanism is responsible for this failure.

5.11 Velocity control

In theory the angular, linear and joint velocities of the end effector can be used to control the PERA. For excerting velocity control in this simulation, the software nodes are dependent on the possibilities offered by the rostopics from the Amigo.

The command *rostopic list* shows that the only accepted velocities are for the joint: joint position, joint velocity and joint acceleration. As a consequence, the velocity control for this thesis can only be the earlier described ξ or $\dot{q}$. Hence, the node publishing the velocities only needs to publish the joint velocities.

5.12 Moving backwards

The Amigo simulation is complex software and the used version is not supported by the Amigo team. Since the start of this thesis, it is noticed that the Amigo is prone to very slowly move backwards, as shown in Table 9. Since with any position the Amigo is in, its back can be in a different direction, this movement is independent of a coordinate. As a result of this backwards move, the working space will change and the object gets out of the reachable working space. This phenomenon is unknown to the Amigo team.

Table 9: Backwards move of Amigo in simulation

time (h)	x (m)	y (m)
0.18	0.60	-2.37
1.34	0.57	-2.365

Because the Amigo is facing the object in the x direction, the backwards move is also in this direction. The shown coordinates are of the `amigo::finger1_-tip_right` link which is a finger tip of the right-side hand.

5.13 Reaching a given coordinate

The first inverse kinematics test in the simulation is done using the inverse kinematics node that comes with the Amigo simulation. An attempt to move the right-side end-effector over a distance of 1 cm after reading the cartesian coordinates of the right hand results in very erratic behaviour such as complete displacement of the robot. Subsequent attempts of changing coordinates, reducing or enlarging the required distance and comparing coordinate systems do not give the desired result. After checking on the source code it turns out that the inverse kinematics algorithm is not fully implemented and does not work. This is confirmed by the Amigo team.

The decision is made to use a Matlab implementation by Dragan Kostíc instead, which is written with the Amigo in mind. After porting the code from Matlab to C++, the kinematics are still wrong. Since the simulation is started with `start.launch`, this file is checked and it turns out that the node `amigo_-velocity_kinematics` might interfere. After disabling this node, the inverse kinematics node seems to work correctly.

The first series of tests is without the use of the DTG algorithm. The objective is to reach the can at the coordinates $(x, y, z) = (1,\ 2.4,\ 0.8)$. To accomplish this, the robot is turned clock-wise over an angle of $0.78rad$ or $45°$ and the maximum linear twist is set at 0.5. From the starting position, the arm starts with a delay of several seconds and then moves to the correct x and z coordinates. The y coordinate is not reached, seemingly because the end effector collides with a table edge. An oddity is also that the orientation of the end effector changes: it tilts to the left and back to horzontal. This is corrected by comparing the orientation of both left and right end effector and adjusting if necessary.

After finding the coordinates of the fully stretched arm using forward kinematics pointed towards the can, it shows that the given final end effector coordinates are actually outside of the reachable workspace of the arm. It also shows that the coordinates of the end effector (and thus of the whole robot) are world coordinates. To solve the workspace problem, two corrections are made. The first correction is that the x coordinate of the can is changed to 0.9 which is the edge of the table. The second correction is that both x and y coordinates as well as the rotation of the Amigo is changed, so that the Amigo faces the table head-on.

After starting the simulation, the PERA moves towards the position where it detects the object. However, not all joint angles are immediately calculated correctly. During the tests it takes up to 6 restarts of the simulation before the initial position is calculated correctly. Since the algorithm by D. Kostić is proven to work in Matlab, it remains unclear why this happens.

5.14 Reaching a detected object

A detected object can be reached by searching for an object with a distinct colour. After that, the length of the contour of this object has to be determined. With the contour length, the ratio between the image contour and object's contour is calculated. Based on this ratio, the speed is determined: the larger the ratio, the faster the speed. As described earlier, the problem with using the contour length is that the contour is not always constant. The use of keypoints has the problem that its number is not constant and they are not equally spread.

As soon as the presence of the object (the can) is detected, the speed is adapted according to the distance. It is clearly visible that the end effector approaches the object, reduces the speed and stops. However, instead of coming to a full stop, the arm withdraws a bit and then comes to a full stop while the nodes keep running. This also happens when the nodes are stopped.

This suggests that there is either a node or some other lower level process working during the running of the nodes. However, when observing a map of ROS activity using `rxgraph` both during and after the running of nodes, no interfering node or process is shown.

During most simulations it is observed that the PERA, and thus the end effector, withdraws after reaching the coordinates in a fading oscillating movement. Since efforts to eliminate this have no effect, the conclusion is that this is a hard-coded movement in the Amigo simulation.

5.15 Observations after tests

A couple of times, after all four nodes are stopped, the PERA moves into a very different position. This happens either shortly or some time after ending the test. This time is not measured. Examples of these positions are

- fully stretched down
- fully stretched horizontally sideways

Unfortunately, it remains unclear what the cause of this behaviour is. The Amigo team declined to help since it was an outdated and thus unsupported version of the simulation. Using a new version of the simulation would at least require an operating system upgrade of the laptop and an upgrade of ROS with a rewrite of software to solve ROS incompatibilities. However, since several observed issues were unknown to the Amigo team, it remains unknown if those would have been solved.

5.16 Conclusion

The objective of the simulations was to test the designed software architecture and modules and find out if these met the expectations.

Tested were both forward and inverse kinematics with and without trajectories, with given and generated trajectories.

The forward kinematics seemed to work well, but the application of inverse kinematics and trajectories did not function at all in the Gazebo simulation. The inverse kinematics from the used Amigo simulation is present in basis, but it is not functional. An inverse kinematics algorithm from dr. Dragan Kostić and the trajectories showed no problem in the Matlab simulation environment.

It turned out that the Amigo simulation had issues such as a continuously moving robot with random erratic behaviour. Since the used version of this simulation went out of support during this thesis and is complicated software, it was hard to determine what the causes of the problems were.

Although the Amigo simulation was used both to save time and to not reinvent the wheel, developing a PERA simulation would probably have been better.

6 Experiments on the physical robot and embedded system

The experiments are done both with the laptop which is used for simulations as well, and on the BBxM. It should be noted that for the simulation as provided by dr. Dragan Kostić, the gravitational acceleration was defined as $g = 0$. However, for the experiments this value can not be used since in the Netherlands on average goes that $g = 9.81m/s$.

Considering the ROS website and various sources on the internet, Ubuntu and Ångström are the two Linux distributions that are most used for ARM-based embedded systems with ROS. The objective of the first experiments is to see which of these distributions can be used on the BBxM to work with both the DSP and ROS.

The use of the BBxM in this thesis was intended to be a small part, but as a result of the ongoing experiments its part gradually became larger.

6.1 ROS on Ångström

In order to get Ångström to run on the BBxM, the file `u-Boot.bin` has to be placed in the boot partition. To ensure the correct use of memory, `uEnv.txt` is necessary as well.

It is available with either console-only or desktop environments XFCE or Gnome. The use of a graphical environment is practical because of the use of the camera to check for the object. Since the use of resources by XFCE is known, and for Gnome is about 500MB, it makes the choice easy.

ROS is available via OpenEmbedded Layer [38] and is cross-compiled using *bitbake* from the BeagleRos project [39]. OpenCV is available via repository.

Although DSPLink is readily available for Ångström, and the communication with the DSP usually works well, testresults using example applications from TI show that its working is not guaranteed. It also shows that even with two successive runs of *./helloDSPgpp helloDSP.out 5*, intending to send 5 messages back and forth between GPP and DSP, the first run goes well and the second run shows (amongst other) a message "DSP-side configuration mismatch/failure". The only way to get the GPP-DSP communication working again, is a restart of the BBxM. When `helloDSPgpp` fails, the other available example application `poolnotify` (which uses the 4096 bits stack) will fail as well and vice-versa. If the communication with the DSP works well, running *helloDSPgpp* gives an immediate result showing that 5 iterations took "0 seconds and 976 microseconds" (time may vary).

Apart from the DSP/GPP communication, ROS has to work on the BBxM as well. To accomplish this, several experiments are done to install ROS on the BBxM. These experiments using the earlier mentioned OpenEmbedded show that ROS is only available when a experimental Linux kernel is used. Any image created with this kernel does not boot the BBxM, despite adjusting the boot partition. Although the support for the OpemEmbedded software indicates that BeagleBoard is supported, this does not include the BBxM. Results of the experiments confirm this. Also the use of an image with a stable Linux kernel shows that it is not possbile to install ROS in the BBxM using Ångström. Since the challlenge of the work is to combine both a DSP and ROS, the experiments continue using Ubuntu.

6.2 ROS on Ubuntu

Since ROS for ARM is listed on the ROS installation page, though experimental, it is assumed that a basic installation and eventual needed dependencies are available from repository.

To use Ubuntu, the last supported desktop version for armhf is 12.04 which comes just like the x86 version with the Unity desktop installed. Since this desktop is a memory hog and slows down all activities, tests have been done with a few other low-memory desktops to free up memory. For optimal use of the memory the following setting and applications were used:

- LXDE [41] desktop which needs 256 MB RAM for minimal installation with graphical environment to have as much memory available as possible for applications and being able to view the camera output.
- FVWM [42] desktop which needs less than 256 MB, but does not handle the mounting of USB devices nicely.

In the end however, to ensure that as much memory is available for this thesis, no desktop environment is used. Instead all activities are done using "ssh -X" sessions so all graphical output is shown on remote desktop.
After the first minimal installation of ROS Groovy and coying the sourcecode of the nodes to the BBxM, building the `arm_image_proc` node shows that quite a few more ROS packages are necessary.

An issue is that ROS Electric uses a different vision library for capturing images than ROS Groovy. So this is changed and due to differences with camera drivers the ROS version on the BBxM is changed to Hydro.

With both a laptop and an embedded system available, several hardware combinations to utilize the PERA are possible. The following experiments are considered to find out which hardware combination can be used:

- using only the laptop and the PERA, which means that both the camera and the PERA will be connected to the laptop.
- using a laptop, an embedded system and the PERA. This means that the PERA will be connected to the laptop and the camera will be connected to the embedded system. The imaging node is running on the BBxM while the other nodes are running on the laptop. The imaging node will use the DSP.
- using "full embedded control": both the camera and the PERA will be connected to the embedded system. Like with the previous option, the imaging node will use the DSP.

The first experiment is carried out without the camera and shows that the PERA can immediately run a homing task from Philips without any problem. An experiment with camera only shows no problem either.

The second experiment can not been carried out. Reason for this is that the ROS versions on the laptop and the BBxM are different and the serial communication node provided by ROS is not available for the version on the laptop. Apart from that, by experts it is highly recommended to not use more than one version of ROS at the same time. Reason for this recommendation is that the messages differ for each ROS version and as such can give problems when messages from one version are misinterpreted by the other.

The third experiment starts with letting the PERA do the homing task. However, instead of a smooth move, the PERA starts to oscillate. The BBxM manual states that all four USB ports have full LS/FS/HS support.

With this last result, the settings of the PERA software are changed to only reading the joint angles. It shows that these can be read out well, but with both a delay and an error message about a failing amplifier board #2. According to the PERA manual, this is the RT-Motion USB #3 board which interfaces both with the shoulder rotation motor and encoder and with the gripper motor and encoder. After reconnecting the PERA to the laptop, it works as before. The suspicion is that there is a timing issue at hand, but since the manual does not provide timing data, this can not be confirmed.

Regarding the connection of the PERA to any computer, the manual of the PERA recommends that no other USB devices should be used at the same time as the PERA. This leads to the experiment of disconnecting the camera, mouse and keyboard from the BBxM after starting the homing task. However, this makes no difference since the PERA starts to oscillate again. The only difference with the first oscillation is that there is no message about the RT-Motion USB board and the oscillation starts at the wrist instead of at the shoulder.

Regarding the use of the USB ports, the manual of the BBxM states that for power-consuming peripherals, a power supply should be used that delivers more than $1.5A$. So the next experiment is to use a power supply delivering $4A$. This makes no difference compared with the previous experiments.

To be sure, the PERA is tested again on the laptop and shows no problem. This leads to the assumption that the USB communication as used by the RtMotionUSB boards is not compatible with the BBxM.

Another considered cause is a difference in endianness. Endianness is the sequence in which the least and most significant bit [7] (lsb and msb) are placed. With little endianness, the lsb is placed at the right-hand side of the number as is the case with the decimal numbersystem.

Intel-based computers are by default little endian, while Cortex-A8-based computers are mixed-endian [40]: instructions are fixed little endian, data access can be set to either little or big endian. This is controlled by the E bit in the Program Status Register. A quick programmed test reveals that the output is little endian. With this result, the endianness is ruled out as a cause.

As discussed previously, to experiment with the DSP, a driver is necessary. The process of creating both the GPP and DSP side of the software does not go smooth, despite the good and comprehensive guide from Texas Instruments. Another issue is that there actually is support for the OMAP3530, but not for the DM3730, probably because both are considered to be more or less equal.

Continuing with the BBxM, the nodes are copied and compiled. It shows that the nodes have many ROS dependencies that are being installed via internet during compilation.

For the following experiments, the imaging sofware will have to be compiled with the special DSP toolset in order to make optimal use of the DSP.

[7] BInary digiT

6.3 DSP driver

The utilization of a DSP is part of this thesis. The objective of these experiments is to test the use of the DSPLink driver using either Ubuntu or Ångström.

For Ångström, the first option is to use the tools from the Embedded Systems group at the Delft faculty of Electrical Engineering, Mathematics and Computer Sciences (EEMCS). Requirements for a successful compilation as stated by them are

- keeping the kernel magic the same
- cross-compiling the toolchain
- using gcc 4.7 for armhf architecture

As it turns out, this software is different in that the Linux distribution they offer is using Ångström kernel 2010.7-test-20110220 and it is designed for the standard BB . As described earlier, running a short `helloDSP` communication test ends after a random number of runs with the message that communication failed. So although the DSPLink driver does work, it is not stable. This is confirmed by Texas Instruments telling that the type of DSP on the BBxM is a different model than on the BB. However, a version of the test for the BBxM is not offered.

The second option is the Narcissus website [43]. Here it is made easy to create Ångström images with DSP. Upon testing though, the driver shows the same behaviour as with the software provided by the EEMCS. After various forum inquiries, it turns out that these images are for the standard board and thus for the OMAP3530. The driver does work, but as before not very stable.

A third option is Circuitco website [44]. This website offers the image as delivered with the BBxM but not the DSPLink driver. Cross-compiling is necessary but very tedious using the same options as at EEMCS:

- Normal: -mcpu=cortex-a8 -mfloat-abi=hard -mthumb-interwork -march=armv7-a -mfpu=vfpv3-d16
- Speed: -mcpu=cortex-a8 -mfloat-abi=hard -mthumb-interwork -march=armv7-a -mfpu=neon -funsafe-math-optimizations
- Safe: -mfloat-abi=hard -march=armv7 -mfpu=vfpv3 geen fpu: -mfloat-abi=softfp -march=armv7

Despite that the compilation goes well, the remaining problem is that ROS has to work as well. And previous experiments on the BBxM with Ångström show that this is a combination that does not work.

The next part of this experiment is to get DSPLink working on Ubuntu. Ubuntu is available in different versions, such as 12.04 LongTermSupport (LTS) and 13.04. Because of the support of the LTS version till 2015, this one is chosen.

For using the DSP, the DSPLink driver has to be built against the Linux kernel, using the toolset offered by Texas Instrumments and the recommended Lite version of the cross-compile toolchain CodeSourcery. After cross-compiling the same kernel version as used on the BBxM and building the DSPLink driver against it, it turns out that the driver is built for the ARMv6 architecture, instead of ARMv7.

One reason turns out to be that although it is recommended to use CodeSourcery, it only compiles for armel (ARM EABI Little-endian), not for armhf (ARM hard-float). For armhf it is recommended that the Linaro toolchain is used. However, after adjusting the toolchain, the driver is still compiles for ARMv6. Searching the configuration files, `autoconf.h` shows that the kernel is prepared for ARMv7 but that an `omap3_beagle_defconfig` configuration file is missing so it cannot be built for ARMv7 in combination with a kernel. Checking various kernels shows that this configuration file is only delivered with the TI toolchain. Attempts to use the configuration file with a current kernel end up without usable result.

Conclusion of this series of experiments is that although various posts on the internet forums show that the driver works fine, this does not apply to the BBxM.

6.4 Real-time

The real-time part of a Linux kernel consists of third-party libraries. The build of the necessary Xenomai libraries succeeds with the help of Paul Corner, who provides a patch for the (until Feb. 2014) unsupported armhf architecture. The armel architecture already was supported.

After patching the kernel, cross-compiling it for Xenomai goes well and a `linux-image` is created. Installation following the normal procedure with the `dpkg` command and subsequent booting results in hanging of the BBxM. The ten next experiments with changes with respect to debugging and display drivers have the same result. Then using a different way of compilation which ends up with a `uImage` and placing it directly in the boot partition results in a working situation.

On recommendation from the Xenomai experts, Xenomai 2.6.3 and its latest supported kernel 3.10.18 is used. With this, the Xenomai kernel boots and the Xenomai tests results are as expected. However, in order to use the PERA, `ROS::master` must be running. This is started with the *roscore* command and although it starts, it does not start the ROS::master. As a result, the PERA can not be activated. Installing ROS Hydro gives the result that roscore starts and gives the message *unable to contact my own server at [http://BBxM:<port>]*, where port is e.g. 11311. This message would be due to networksettings, but these are correct. Further tests show no improvement.

Another option is disabling the graphical environment on the BBxM and only have graphics via `ssh -X`. Although this works and xterm can be used, the working of roscore does not improve. See table 10. For a comparison, kernels 3.2.51 and 3.10.18 also are installed on a computer with a 2 GHz AMD AthlonXP 2400+ CPU and 2 GB RAM, with the ability to boot the 3.10.18 kernel with and without Xenomai. On this computer, no problems are found.

In order to find out possible causes, the following experiments are done:

Testing and possible replacement of the Python version, since `roscore` is a Python script.
Result: *valgrind roscore* shows errors on both the testmachine and the BBxM

Table 10: Results of starting roscore and ros master

kernel	xenomai	roscore via ssh	roscore via ssh -X
3.2.51	n	y	y
3.10.18	n	n	n
3.10.18	y	n	n

while the roscore problem only occurs on the BBxM. So this can not be the cause of the problem.

Reading from `/proc` by *top* and *ps* to find a difference between testmachine and the BBxM.
Result: on the BBxM *top* stalls while *ps* works correctly, on the testmachine everyone works as expected. Running *busybox top* shows no difference but the use of *busybox* excludes version mismatches and library issues. However, it does not reveal a real cause of the problem.

Running *strace -p <roscore pid*[8]*>* to find out which pid is running last.
Result: it becomes clear that a *futex* systemcall, which is a fast userspace locking mechanism, is the last action before stalling. *strace* stalls an indefinite time, until *Ctrl-C* is pressed. *roscore* still runs until the pid is killed. Like the previous test, it does not reveal the cause.

Testing Xenomai on the 3.10.18 kernel. Although the patch for this kernel is available for both the ARM and the x86 architecture, the developers state the patch is not officially released and thus not supported. Despite of this situation, an issue with transition to cpu-idle mode was fixed with a patch that is made available. However, since the running of *roscore* is not dependent on Xenomai, the patch is not applied.

Using ROS Hydro on kernel 3.10.18 to find out if this combination works.
Results: *roscore* starts once including the ROS::master. However, when starting the Orocos wrapper, errors appear that stack/packages can not be found. This can be solved using *rosws set <name of package>*, but this is a ROS Hydro solution only and therefore not further investigated.

By incident, checking the time shows the following. The *date* command shows that the systemdate and time on the BBxM runs fine on kernel 3.2 but does not on kernel 3.10.18: time stops running after boot. It turns out that the `OMAP_MCBSP` option for enabling the clock is missing from the 3.10.18 kernel. Without the time running, real-time activity is impossible. It turns out that `mcbsp` as part of `SND_OMAP_SOC_MCBSP` is relocated to the `alsa` sound. After enabling this kernel-option, testing shows that both the network and USB fail for the 3.10.18 kernel. Reverting the `MCBSP` setting does not re-enable the network and USB. However, with kernel 3.14 configured with the configuration used for 3.10.18, it shows that the network works and *roscore* and *top* run fine. Drawback is that there is no Xenomai patch for the 3.14 kernel yet. Further tests show that while package management was an issue on the 3.10.18 kernel, it works fine on the 3.14 kernel.

According to the ROS site, ROS Groovy is supported up till Ubuntu 12.10 Quantal which runs a kernel from the 3.5 series. So the next experiment is to try kernel 3.5.7, as this version is also supported by the same Xenomai version.

[8]Process ID, a number assigned to a running system process or activity

Results: the network does not function which is indicated by the lights being off. Also both the mouse and keyboard are unresponsive. According to [45] this is due to a kernel bug that causes EHCI USB to fail. Following a recommendation to update the bootloader `u-boot.bin` according to [46] the result is a boot ending with a kernel panic and the message *VFS: Cannot open root device "ubi0:rootfs" or unknown-block(0,0): error -19.* After examining the serial console output, it turns out that the boot sequence wants to boot from a zImage kernel instead of uImage. After adjustment to boot from a zImage, there is no improvement so the 3.5.7 kernel is of no use.

The next experiment is to see if the last kernel supported by this Xenomai version will run on the BBxM: the 3.8.18 kernel.
Results: the network does work in such a way that an IP address is requested. However, establishing an ssh session is denied and a *ping* command receives no response. The video stays black with the message that the 19 inch monitor is unable to handle the resolution delivered by the TFP410 videochip of the BBxM. Because of the EHCI patch experience with the 3.5.7 kernel, the result for this kernel is expected the be same so the 3.8.18 kernel is of no use.

The latest kernel being used for this thesis is 3.14 but since the still ongoing development of the Xenomai patch during the final stages of this thesis it is not tested.

6.5 The Eigen library

For the use of both the Direct Trajectory Generation algorithm and the inverse kinematics algorithm, the implementation is adapted for use with ROS Hydro on the BBxM. Reason for this is that ROS Electric, which is used in the simulation, is the last version for which the Eigen functions are native to ROS. Since then, the Eigen library is an operating system dependency and as such has to be installed by the operating systems' package manager. The adaptation is a change in the `manifest.xml` file of both the DTG and the inverse kinematics node.

6.6 Camera behaviour

The following experiments with the camera are done to see if and how a camera can be used for this thesis.

Earlier in this thesis the option is mentioned to mount a camera like the LI-5M03 directly on the BBxM. For this thesis, direct mounting of this camera requires the BBxM to be located on the end effector, which is not practical considering its interfaces. The solution is to use a cable to overcome the distance between the BBxM and the camera when it is located on the end effector. However, the necessary driver is only supported for the 2.6 kernel used by Ångström. Considering the results of the experiments with ROS on Ångström, this is no solution for this thesis.

The Logitech camera used for this thesis, is tested using both laptop and the BBxM with a variety of software. On the laptop, the cameratools GUVCView and Cheese are used to find its specifications and possible speeds of capturing images. For its use with ROS, the `gscam` driver is used.

On the BBxM, the aforementioned cameratools are used, as well as 3 ROS camera drivers that fit the used ROS versions. Also, to find the effect of memory use, various desktop environments are used: fvwm, LXDE and XFDE. In the end, the desktop environments are all discarded because of the use of graphics via *ssh -X*.

Reasons for using the different environments is to find out the difference in support for drivers between the releases and the difference in support between the x86 and ARM platforms.

Because of the support of drivers by Ubuntu versions 12.04 and 13.04, both have been tested. The final result is that version 12.04 is used for most experiments. The results with `12.04` are described next.

In order to be able to see what the camera captures, there are two applications being recommended: `GUVCview` and `Cheese`. These application will not be used by the rosnodes because the gscam rosnode will be used.

As expected, on the laptop there is no problem at all. With both applications all moves in front of the camera were shown on screen without delay.

The results on the BBxM are different. Since Cheese works well on Ångström, but crashes on Ubuntu 12.04 and GUVCView works well on both, the latter is chosen.

Using GUVCView it shows that while the frames-per-second (FPS) settings at 30 with a resolution of 640×480 are the same, a relatively slow movement in front of the cam goes fine, but a fast movement shows a delay of about 1 second. Since the homing test shows that the PERA can move quite fast, and the response has to be made in real-time based on the captured images, this can be a real problem. Therefore, the velocity of the PERA has to be limited so the images are shown on screen in real-time. Also, the images must be grabbed at high speed. In the setup, GUVCView can be set to a net frequency of either 50 or 60 Hz. Changing from the standard of 60 Hz to 50 Hz does not improve the capturing response.

While on ROS Electric the `gscam` driver is used, on ROS Groovy and Hydro this driver does not work. On both the BBxM and the comparison machine, attempts with `gscam` result in error followed by a core dump. After consulting ahendrix (ROS armhf developer) the `uvc_camera` driver for Hydro is used instead. After changing the camera permissions to 444 (rw.rw.rw.), it runs as a separate node and provides input for the image processing node. Since the simulation uses Electric, 2 nodes BBxM need adaptation for ROS before running on the BBxM. This adaptation shows that in some cases there is no backwards compatibility with ROS. This concerns the Eigen library, used for matrix calculations in C++ which needs a change in the `CMakeLists.txt` and the imaging code which needs a complete rewrite.

For Hydro, the camera node uses the `uvc_camera` driver which by default uses 2 cameras. In order to use only 1 camera, the option `uvc_camera_node` is used. It shows that the camera calibration is not always good. In such a case very vague and/or very bright images are streamed. The solution is to restart the camera node until the streamed images are good.

For completeness, the results with version `13.04` are described next. It turns out the situation for viewing the camera is the other way around. `GUVCView` meets with an malfunctioning ALSA installation while `Cheese` does start. Started from a terminal window error messages *failed to create a pipe screen for omap*, *failed to open omap* and *Can't record audio fast enough.* The image refreshrate in the Cheese window is very slow, and with the resolution set to 640×480, the capture rate in *burst mode* can not be set faster than with a delay of 1 image per second. Since this is too slow, the camera can not be used. Testing this with on a regular computer, there is no problem with GUVCView.

6.7 Latency of the camera

The images taken by the Logitech camera are shown with a latency. There can be several reasons for this.

The net frequency can be of influence with respect to the fps. In *simulation*, there is only latency caused by the modelling-induced fps. In *reality*, if the net frequency is not the same as the frame rate, black images may occur in the output. In this thesis, this has not been experienced.

The frame rate of the real camera may differ from that of the simulated camera. The simulated camera used in this thesis, is set to 30 fps. Using the GUVCView tool, the fps of the real camera can be set at various values. For the libuvc_camera driver, this value is set in the launchfile and the fps shows no issue.

The computer properties can be major influences since image processing is dependent of the amount of system memory (RAM), the CPU speed, the used vision algorithm and the availability and use of digital processing hardware such as a DSP. As described earlier, the DSP is not used in this thesis. Since the RAM and CPU speed can not be changed, the only variable for optimization is the vision algorithm. For this, OpenCV is used.

The use of real-time software. While this can be expected, it shows that the latency decreases while using a Xenomai-patched kernel on the laptop. Experiments on the BBxM have not been done due to the earlier described issues on real-time.

6.8 Use of the OpenCV API

The simulation is written on Ubuntu 10.04 while on the BBxM Ubuntu 12.04 is used. For the image processing node, this has consequences. When compiling the node on the BBxM, a couple of OpenCV instructions compile with the error that the instruction does not exist. All of the instructions are part of the C-API and have to be replaced by a C++-API version. However, as the C++-API instructions contain different arguments, this has as a consequence, that both the image capturing and the colour filter for the image segmentation require a rewrite.

6.9 Imagestream via remote shell

After the experiments with the camera on the BBxM in combination with the various desktop environments, it is decided to test remote graphics by *ssh -X* and using the uvc_camera driver. This makes that any graphic application started on the BBxM is shown on the remote desktop.

After the node is started, the stream window pops up immediately. Then it takes at least 5 seconds before the stream is shown and the stream frequently stops after the first image. When this happens, a restart of the node is required after which the image does stream correctly but with a delay up to 5 seconds.

This variable delay can be due to a couple of reasons. One reason can be the remote (ssh) connection that is used. Large amounts of data have to be sent over the network which takes time. The variation in time is then due to the variation in data. Another reason can be the relatively low CPU speed and use of RAM by the graphical process on the BBxM but this should be constant.

To find out which lighting source has to be used for the best filtering result, tests are done with a lamp as well as daylight. The result of using direct daylight is a blank imagestream due to overexposure of the camera. It turns out that the uvc_camera driver applies hardcoded maximum values for brightness and contrast. As a result, the captured images are hardly usable.

6.10 HSV Colour values

Part of the replacement of the OpenCV instructions as described earlier is the *cv::inRange* function. This function uses HSV instead of BGR. HSV stands for Hue, Saturation and Value while BGR stands for Blue, Green, Red. A colour that is detected in BGR can be converted to HSV and vice-versa.

In order to make sure which colour values to use with the *Scalar* argument in the OpenCV *cv::inRange* function, pictures of test objects of roughly the same colour are made and the HSV colour values measured using Gimp. Using these values in OpenCV will give problems since according to [47], the range of HSV values of Gimp differ from those of OpenCV as shown in Table 11.

Table 11: Difference between Gimp and OpenCV HSV values

	H	S	V
Gimp	0 - 360	0 - 100	0 - 100
OpenCV	0 - 180	0 - 255	0 - 255

The probed Gimp HSV values of one of the pictured objects, a yellow tennisball are 74-47-55. Converting the values to OpenCV gives the values in Table 12.

Table 12: Conversion between Gimp and OpenCV HSV values

Gimp	factor	OpenCV
74	$\div$ 2	37
47	$\times$ 2.5	117.5 $\approx$ 118
55	$\times$ 2.5	137.5 $\approx$ 138

To create the lower and upper boundaries for the *cv::inRange* function, subtract or add 20. Another test using a bordeaux-red mobile phone shows that it is well detected by the camera and that filtering works. Detecting its contours however, shows a difference between simulation and reality. While in simulation an object shows just as it is, in reality light can cause an object to show on camera in different shape. After processing the captured image of the phone, it is not shown not in its entirety. As a result, correctly determining the distance to the object is difficult.

6.11 The libuvc_camera driver

Because of the hardcoded settings of the uvc_camera driver, a fork is created by Ken Tossell. This is the libuvc_camera, providing almost the same ROS topics as uvc_camera but it does not have hardcoded settings. Instead, the settings need to be provided in a launch file. The driver also works faster than uvc_camera. To make it work, instructions from [48] must be applied.

Testing using the libuvc_camera driver with the same mobile phone as an object using a normal lamp, the distance can not be determined from the captured image as its size is undeterministic. Using daylight, the size can not be determined by its contour either.

Using a yellow tennisball as an object, the filtering by daylight shows to be much better than when using a lamp. The contour by daylight is better as well although not useable either because only a small part of the tennisball is clearly contoured. Most of the edge is faint and interrupted.

Without real-time implemented, and although this driver is faster than the uvc_camera driver, the time it takes to show an image in the window is more than 1 minute. Once the image is shown, it is updated every 3 to 5 seconds. Since real-time is not implemented yet by lack of the patch, it can not be compared with a real-time situation.

6.12 Source of light

As simulation shows, determining the contour and arclength prove only possible by approximation. The experiments are done to see results from using a lamp at a netfrequency of 50 Hz as well as daylight.

The results of using a lamp are that images are not constant. Reflections and saturation both influence the edges and the form of the object so both contour and keypoints methods are of no use. The results of using daylight are better but still make determination difficult.

6.13 Driver for the PERA

To be able to use the PERA, a driver is needed to get access to it. Proof of a successful load of the driver into memory is the creation of a file called `/dev/RTMotionUSBx` where `x` can be either `a` or `b`. To accomplish this, a load script is available.

The driver `rtmotion_usb.c` is originally written by Philips Applied Technologies and when compiling it, several error messages are shown. These point to the fact that the driver is created for Linux kernels up to 2.6.35 and as such is no longer compatible with modern kernels.

In order to make the driver work with kernel version 3.2, buffer and error definitions in the driver source code need to be replaced. For kernel versions 3.10 and newer, `proc_` functions have changed.

The USB bus number is important as well. It indicates the bus to which the PERA is attached. This must be determined prior to compilation, so the PERA has to be attached to the USB in power-on mode and checked with the command *cat /proc/RtMotionUSB*. This shows that for the BBxM the file `src/IoConf.c` has to be changed. The shoulder, upper arm, elbow and wrist each have their own definition.

After adapting the driver and starting a homing test, it shows that the PERA runs well on the laptop while it starts to oscillate immediately on the BBxM. This can be caused by two reasons. The first reason is the fact that both the network connection and the USB-A ports are connected to the USB bus. These signals may interfere. The second reason, which is more plausible, is that during the these tests, Xenomai is not running.

On another test using Ubuntu 13.04 with kernel 3.12.0, the driver does not compile giving errors `implicit declaration of function ‘create_proc_entry’ [-Werror=implicit-function-declaration]` and `dereferencing pointer to incomplete type`. The first error is found back in a bugreport for kernel 3.11.0 and is reproduced on a normal computer. As a result, this combination of kernel and driver can not be used and Ubuntu 12.04 is restored.

6.14 Serial and USB interfaces

The BBxM has RS232, USB-A and USB-OTG interfaces. During the experiments it became clear that the use of each interface is not just driver dependent but also kernel dependent.

The possiblity of using the RS232 interface as a full terminal only works when using an old kernel such as 2.6 with for example Ångström. Reason for this limitation is that the kernel-option of low-level UART [9] debugging is removed from later kernels.

When both the RS232 and ssh connection are used, it shows that the RS232 connection is necessary. Reason for this is that when the BBxM is shutdown, the ssh connection is the first connection that is closed by the BBxM. The RS232 connection shows the entire process of shutting down until the message that the BBxM has halted.

With kernel 3.2, the RS232 interface can only be used to watch the boot process until the kernel is loaded into memory. In order to watch this happen, the `minicom` program is used with `ttyUSB0` as interface selected. Any input can be recorded.

The USB-OTG interface can be used to connect the BBxM as a client to a host computer via USB-over-ethernet. After connecting and checking with the command *ifconfig -a*, the `usb0` interface should show up. Despite attempts this connection proves to be unable to work on Ubuntu 12.04.

Using Ångström, the ethernet-over-USB connection via USB-OTG proves to be not working either. Although an ssh connection can be established, the password is not accepted. As a result, this interface is not usable.

6.15 PERA joints vs. links

Another difference between the simulation en the PERA is in the topics. In the simulation, the cartesian link coordinates can be read out from the topic gazebo_msgs::LinkStates. This way, it can be checked if the end-effector of the PERA has reached its destination. With the PERA, the topic equivalent is amigo_msgs::arm_links. It can not be used in a similar way though, because in can not be instantiated to for example the end-effector. The topic used by the Amigo is amigo_msgs::arm_joints as only forward kinematics is used by that

[9] Universal Asynchronous Receiver Transmitter

project. In this way, checking if the end-effector has reached the desired or calculated cartesion coordinates is not possible.

6.16 Conclusion

The objective was to use an embedded system for partial or full control of the PERA with the use of ROS and the DSP. The choice for the BBxM was given by the combination of a DSP for image processing, sufficient RAM and the availablity of network and USB interfaces. Although the BBxM seemed to be a good choice given its properties, the combination with software was not optimal.

Low-cost, off-the-shelf embedded hardware comes with limitations. In the case of the BBxM, such a limitation is the use of the SMSC LAN9514 chipset which connects the USB bus and ethernet where ethernet is actually simulated by ethernet-over-USB.

Tested were combinations of the Linux distributions Ubuntu and Ångström with the DSP driver, camera drivers, ROS and the real-time kernel-patch Xenomai. It turned out that the use of ROS is mutually exclusive with the DSP driver where it comes to support by the Linux kernel.

The one-camera-at-one-position method as described in section 2.2.5 shows to be not accurate enough for velocity control. For stereo-vision, the end effector is not large enough to hold 2 cameras of the used model and it remains to be seen if the BBxM is capable of handling the simultaneous input from 2 cameras.

The use of open-source software is both an advantage and a disadvantage. The advantage is that it is easy to adapt, free to use, and communities are ready to help you. The disadvantage is that the update-rate of additives may be different and does not necessarily keep pace with eachother and or/and an operating system. As a result, they may not be compatible.

For commercial software the ”time-to-market” is essential and delay is usually not allowed. For open-source software the stability of the software is usually the most important. Most of the time open-source software, be it an operating system, application or a driver, is only released when it is considered to be stable enough. In other cases, releases are as-is. In the case of Xenomai for the 3.14 kernel, timing issues are the reason it is not released at the end of this thesis.

7 Conclusions

7.1 Simulation

The objective of the simulations was to test the designed software architecture and modules and find out if these met the expectations.

Tested were both forward and inverse kinematics with and without trajectories, with given and generated trajectories.

The forward kinematics seemed to work well, but the application of inverse kinematics and trajectories did not function at all in the Gazebo simulation. The inverse kinematics from the used Amigo simulation is present in basis, but it is not functional. An inverse kinematics algorithm provided by dr. Dragan Kostić and the trajectories showed no problem in the Matlab simulation environment.

It turned out that the Amigo simulation had issues such as a continuously moving robot with random erratic behaviour. Since the used version of this simulation went out of support during this thesis and is complicated software, it was hard to determine what the causes of the problems were.

Although the Amigo simulation was used both to save time and to not reinvent the wheel, developing a PERA simulation would probably have been better.

7.2 Embedded System

The objective was to use an embedded system for partial or full control of the PERA with the use of ROS and the DSP. The choice for the BBxM was given by the combination of a DSP for image processing, sufficient RAM and the availablity of network and USB interfaces. Although the BBxM seemed to be a good choice given its properties, the combination with software was not optimal.

Low-cost, off-the-shelf embedded hardware comes with limitations. In the case of the BBxM, such a limitation is the use of the SMSC LAN9514 chipset which connects the USB bus and ethernet where ethernet is actually simulated by ethernet-over-USB.

Tested were combinations of the Linux distributions Ubuntu and Ångström with the DSP driver, camera drivers, ROS and the real-time kernel-patch Xenomai. It turned out that the use of ROS is mutually exclusive with the DSP driver where it comes to support by the Linux kernel.

The one-camera-at-one-position method as described in section 2.2.5 shows to be not accurate enough for velocity control. For stereo-vision, the end effector is not large enough to hold 2 cameras of the used model and it remains to be seen if the BBxM is capable of handling the simultaneous input from 2 cameras.

The use of open-source software is both an advantage and a disadvantage. The advantage is that it is easy to adapt, free to use, and communities are ready to help you. The disadvantage is that the update-rate of additives may be different and does not necessarily keep pace with eachother and or/and an operating system. As a result, they may not be compatible.

For commercial software the "time-to-market" is essential and delay is usually not allowed. For open-source software the stability of the software is usually the most important. Most of the time open-source software, be it an operating system, application or a driver, is only released when it is considered to be stable enough. In other cases, releases are as-is. In the case of Xenomai for the 3.14 kernel, timing issues are the reason it is not released at the end of this thesis.

7.3 Final conclusion

The challenge of this thesis was to see which problems occur when an embedded system is applied to a vision in the loop robotic system and if an off-the-shelf computer is necessary.

Considering the outcome of the simulations and experiments, there are problems that have to be solved so the question if an off-the-shelf computer is necessary can not be answered at this time.

8 Recommendations

8.1 Connectivity

When the BBxM will only be used for capturing and processing the imagestream, the only option for a direct data connection would be the RS232 serial connection. To achieve this, both host and the BBxM must at least run the same ROS version. Since the OMAP-serial option was removed from the vanilla kernel after version 2.6, this kernel is recommended then.

8.2 DSP camera

When the BBxM will be used for full control, the direct-to-DSP camera could be used in order to get faster processing. However, as this requires the by now obsolete proprietary DSP driver, this will require the use of the 2.6 kernel.

8.3 Simulation

In case of a simulation, it would be better to use a simulation from scratch so there both kinematics and control can be designed as necessary and the similarity between simulation and PERA is better.

9 Future work

9.1 Embedded control

Depending on the specifications such as RAM and CPU, full control by an embedded system is yet to be seen. If full control is not possible, considering the varying support by different kernel-versions of USB-OTG, RS232, bluetooth, zigbee (IEEE 802.15.4 standard) or wifi, the best connection with a host is yet to be tested.

9.2 PERA driver and Xenomai

The PERA driver is adapted for use with the newer kernels, but has yet to be thoroughly tested.

Since Xenomai has yet to be released for the correct working kernel, it is also not tested with the PERA driver. This remains to be done.

9.3 Reduce imaging time

For an as yet unknown reason, the time it takes for a captured stream to be shown is with more than a minute too long. For a good working situation, this time should be reduced to a couple of seconds at maximum.

9.4 Algorithm

Since the testing of the algorithm has not been conclusive during this thesis, this needs to be done.

9.5 Embedded hardware

The BBxM is not the most widely used off-the-shelf embedded board. As DSP integration is hard to accomplish on this board, other embedded systems with or without DSP may be better suitable.

10 Acknowledgements

This very challenging thesis has been accomplished with the help of the following people:
prof.dr. Robert Babuška, for being the responsible professor for my faculty
prof.dr.ir. Pieter Jonker, who offered this thesis
dr.ir. Roel Pieters
Berk Çalli, MSc
dr.ir. Wouter Caarls,
ir. Janno Lunenburg, for assistence on the Amigo simulation
dr.ir. Dragan Kostić for his help on inverse kinematics
my parents, for their never-ending support

For support via internet I wish to thank:
Robert C. Nelson,
Paulo "ppisati" Pisati (Canonical)

Gilles Chanteperdrix (Xenomai)
Paul Corner

References

[1] http://www.robocup.org/

[2] Baturone, I. ; Univ. de Sevilla, Sevilla ; Moreno-Velo, F.J. ; Blanco, V. ; Ferruz, J., Design of Embedded DSP-Based Fuzzy Controllers for Autonomous Mobile Robots, Industrial Electronics, IEEE Transactions on (Volume:55 , Issue: 2), feb. 2008, p. 928 - 936

[3] Yang, Simon X. ; Sch. of Eng., Univ. of Guelph, Ont., Canada ; Hao Li ; Meng, M.Q.-H. ; Liu, P.X., An embedded fuzzy controller for a behavior-based mobile robot with guaranteed performance, Fuzzy Systems, IEEE Transactions on (Volume:12 , Issue: 4), aug. 2004, p. 436 - 446

[4] Gohiya, C.S. ; Sch. of Electron., Devi Ahilya Univ., Indore, India ; Sadistap, S.S. ; Akbar, S.A. ; Botre, B.A., Design and development of digital PID controller for DC motor drive system using embedded platform for mobile robot, 2013 IEEE 3rd International Advance Computing Conference (IACC), 22-23 Feb. 2013, p. 52 - 55

[5] Bernhard Rinner, Martin Schmid and Reinhold Weiss, Institut für Technische Informatik Technische Universität Graz, Austria, Rapid Prototyping of flexible Embedded Systems on multi-DSP Architectures, Proceedings of the Design,Automation and Test in Europe Conference and Exhibition (DATE'03)

[6] Jicheng Chen, Qingdong Yao, Peng Liu, Ce Shi, Department of Information Science and Electronic Engineering. Zhejiang University. Hangzhou, MDl6: DSP with Some RISC Features for Embedded System, ICSP'O4 Proceedings, p. 144 - 147

[7] François Chaumette and Seth Hutchinson, Visual Servo Control Part I: Basic Approaches, IEEE Robotics & Automation Magazine December 2006

[8] Jwu-Sheng HuChang; Jwu-Jiun Yang; Jyun-Ji Wang; et all, FPGA-based embedded visual servoing platform for quick response visual servoing, 8th Asian Control Conference (ASCC), 2011, p. 263 - 268

[9] Do Hyoung Kim, Do-Yoon Kim, Hyun Seok Hong and Myung Jin Chung, An Image-Based Control Scheme for an Active Stereo Vision System, Dept. of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, Republic of Korea, Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems September 28 - October 2, 2004, p 3375 - 3380

[10] M. Litzenberger, C. Posch, D. Bauer, A.N. Belbachir, P. Schön, B. Kohn, and H. Garn, ARC Seibersdorf research GmbH, Embedded Vision System For Real-Time Object Tracking Using An Asynchronous Transient Vision Sensor, 4th Digital Signal Processing Workshop, 12th - Signal Processing Education Workshop, 24-27 Sept. 2006, p. 173 - 178

[11] Chunhong Zheng, Yuxin Su and Peter C. Müller; Online Smooth Trajectory Generation for Industrial Mechatronic Systems; Proceedings of the 2007

IEEE International Conference on Mechatronics and Automation, pg. 861 - 866, August 5 - 8, 2007, Harbin, China

[12] Roel Pieters et all, Direct Trajectory Generation for Vision-Based Obstacle Avoidance, 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012,Vilamoura, Algarve, Portugal, October 7-12, 2012

[13] Torsten Kröger, IEEE Transactions On Robotics, Vol. 27, No. 5, October 2011

[14] Vivien Delsart, Thierry Fraichard and Luis Martinez, Real-time trajectory generation for car-like vehicles navigating dynamic environments, IEEE International Conference on Robotics and Automation Kobe International Conference Center Kobe, Japan, May 12-17, 2009

[15] Chi-Kin Lai, Lone, M., Thomas, P., Whidborne, J., Cooke, A., On-board trajectory generation for collision avoidance in unmanned aerial vehicles, Simulation & Control Group, Cranfield Univ., Cranfield, UK, 2011 IEEE Aerospace Conference, 5-12 March 2011, p. 1 - 14

[16] http://gazebosim.org

[17] http://www.ros.org/wiki/fuerte/Installation/Ubuntu, visited 28-11-2012

[18] http://www.willowgarage.com

[19] Robotics Modelling, Planning and Control, B. Siciliano et all, chapter 3.7, pg. 139, equation 3.89.

[20] Robot Modeling and Control, p. 93, Mark W. Spong, Seth Hutchinson, M. Vidyasagar, ISBN 978-0-471-64990-8

[21] R.S. Pieters, Direct Methods for Vision-Based Robot Control

[22] Speeded-up Robust Features (SURF), H. Bay, A. Ess, T. Tuytelaars and L. Vangool, Computer Vision and Image Understanding, 110(3):346 - 359, 2008

[23] Robotics Modelling, Planning and Control, B. Siciliano et all, chapter 3.1, pg. 106.

[24] http://docs.opencv.org/doc/tutorials/features2d/feature_detection/feature_detection.html, visited 29-09-2014

[25] Embedded System Design, A Unified Hardware/Software Introduction, Frank Vahid/Tony Givargis, ISBN 9780471386780

[26] http://en.wikipedia.org/wiki/Atmel_AVR, visited 25-09-2014

[27] Computer Architecture, A Quantitative Approach, Fourth Edition, pg. 66 - 141, John L. Hennessy and David A. Patterson, ISBN 9780123704900

[28] http://www.android.com/

[29] www.ubuntu.com/, visited 28-11-2012

[30] http://www.angstrom-distribution.org/

[31] Can Android be used for Real-Time purposes?, Luc Perneel, Hasan Fayyad-Kazan, Martin Timmerman, International Conference on Computer Systems and Industrial Informatics (ICCSII), 2012, p. 1 - 6.

[32] http://en.wikipedia.org/wiki/Real-time_operating_system, visited 1-10-2014

[33] http://www.xenomai.org/index.php/ , visited 28-11-2012

[34] http://opencv.org/

[35] https://www.leopardimaging.com/

[36] http://processors.wiki.ti.com/index.php/Category:DSPLink

[37] Direct Methods for Vision-Based Robot Control, Application and Implementation; Roel Pieters, 2013; ISBN 978-94-6191-648-8

[38] http://openembedded.org/wiki/Main_Page

[39] https://bitbucket.org/cylonlabs/beagleros/wiki/Install

[40] http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0344k/Cdffdceb.html

[41] http://www.lxde.org/

[42] http://fvwm.org/

[43] http://narcissus.angstrom-distribution.org/

[44] http://circuitco.com/support/index.php?title=Main_Page

[45] http://patchwork.ozlabs.org/patch/257649/

[46] http://eewiki.net/display/linuxonarm/BeagleBoard#BeagleBoard-Bootloader:U-Boot

[47] http://www.instructables.com/id/How-to-Track-your-Robot-with-OpenCV/step17/

[48] http://wiki.ros.org/libuvc_camera, visited 25-09-2014

[49] http://docs.opencv.org/doc/tutorials/imgproc/shapedescriptors/moments/moments.html