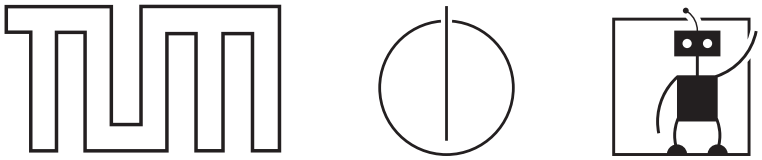


Technische Universität München

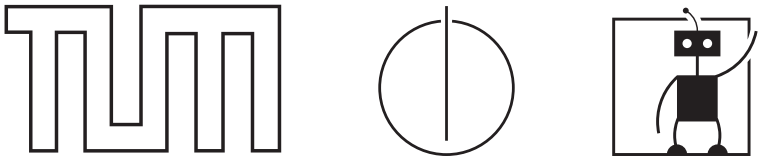
Department of Computer Science

Chair of Robotics and Embedded Systems

Potential field based position control for Mitsubishi RV-6S industrial robots

Bachelor's Thesis

by Matthias Hellerer



Technische Universität München

Department of Computer Science

Chair of Robotics and Embedded Systems

Potential field based position control for Mitsubishi RV-6S industrial robots

Author: ..... Hellerer, Matthias

Supervisor: ..... Prof. Knoll, Alois

Advisors: ..... Dipl.-Inf. Müller, Thomas

Prof. Knoll, Alois

Title: ..... Potential field based position control for Mitsubishi RV-6S industrial robots

Title (german): ..... Positionskontrolle auf Basis von Potentialfeldern für Mitsubishi RV-6S Industrieroboter

University: ..... Technische Universität München

Department: ..... Department of Computer Science

Chair: ..... Chair of Robotics and Embedded Systems

Submission Date: 2009-07-15

Abstract

Manipulator control based on the artificial potential fields method has been shown to be a good solution for real-time obstacle avoidance and navigation in complex, dynamic environments. This paper presents an implementation of this approach for a Mitsubishi RV-6S industrial robot with an emphasis on the use for obstacle avoidance. The robot is able to perform simple navigation tasks while avoiding to crash itself, its static environment or dynamic objects like a person entering its operating space.

Affirmation in lieu of an oath

I hereby declare that I have written this Bachelor's Thesis on my own and have used no other than the stated sources and aids.

Munich, 15.05.2009

.....

Author (Matthias Hellerer)

Contents

List of Figures

A robot may not injure a human being, or, through inaction, allow a human being to come to harm.

The First Law of Robotics
by Isaac Asimov

1 Introduction

The classic approach to robot movement is based on a trajectory for the robot to follow. The trajectory calculation is a computational complex task, requiring significant time. To avoid previously unknown and non-predefined moving obstacles the trajectory has to be recalculated constantly. During the time the recalculation requires the robot moves along the previous calculated trajectory based on an outdated image of the environment.

To overcome this limitation Ossuma Khatib, in his 1985 paper 'Real-time obstacle avoidance for manipulators and mobile robots'[6], introduced a new approach for real-time obstacle avoidance utilizing artificial potential fields. Since then this new approach has been greatly improved and successfully been implemented either as fast low level obstacle avoidance system in combination with a slow, complex high level trajectory planning algorithm or as a complex motion planning algorithm of its own.[5][6]

The following implementation is a hybrid of those two. It may either be used for simple navigation tasks as well as as part of a more complex system. It is designed as a general purpose library for further use at the chair of robotics and the Sonderforschungsbereich[1] 453. Besides this, it has also been used for a concrete implementation in a test setup.

This thesis has been developed as part of the SFB 453 'High-Fidelity Telepresence and Teleaction', more precisely subproject T4. The objective of this project is the development of a telepresence and teleaction system. A system like this allows for a an operator to feel present in a distance location and be able to actively intervene at it. The project is especially focused on minimally invasive, robotic surgery.

[1] special research field

Subproject T4 'Project Area T: Transfer Range - Automated, Robot-Assisted handling of Limp and Deformable Objects' is focused on transferring research results obtained by subproject I4 'Project Area I: Integrated Applications of the HVA-work space - Shared Control for Cooperative Tele-Manipulation in Robotic Surgery: Methods, Implementation and Evaluation' to industrial applications. Its primary field of interest is the handling of highly deformable, limp objects, such as cables, tubes or threads[1].

The subsystem developed for this thesis is built to support the automated handling of limp objects as well as manipulator control by a teleoperator. As proofed by Cahyadi et al. [3] for mobile robots, potential fields are very well suited for the usage in telepresence systems.

This subproject has been conducted in cooperation with Mitsubishi Electric as industry partner. Therefore Mitsubishi Electric loaned the industrial robot RV-6S to the Technische Universität München, which was used for development and in the test setup.

For the purpose of this thesis a test setup has been deployed at the Technische Universität München. It has been used during the development phase and for experiments to prove the practical functionality.

The paper is organized as follows: In chapter 2 related work is introduced and roughly compared to approache shown here. In chapter 3 potential fields are defined and explained. In chapter 4 the environment used for development and experiments is shown. In chapter 5 the control system is explained and finally in chapter 6 different experiments are performed to demonstrate the functionality.

There's a good reason why nobody studies history, it just teaches you too much.

by Noam Chomsky

2 Related Work

2.1 Potential Fields for Manipulator Control

The potential field based approach to motion planning has been around for about 25 years now. As mentioned in the introduction, this research field was founded in 1985 when Ossuma Khatib published his paper "Real-time Obstacle Avoidance for Manipulators and Mobile Robots". Ever since a lot of contributions have been made, greatly improving its capabilities. Yet, the vast majority of these improvements have been made in the field of mobile robots path planning. Even so the paper by Ossuma Khatib focuses on manipulators and only mentions a possible applicability for mobile robots, advances in the field of articulated arm control are very rare. Most findings made with mobile robots have not been back ported to manipulator control.

While developed mainly independently, the techniques introduced here show a lot of similarities to the work by Ossuma Khatib, yet two important differences are worth mentioning: Khatib uses Lagrangians for converting forces, calculated using the artificial potential field, into a movement of the manipulator, while the approach demonstrated in 5.1 is based on mechanics and the exertion of mechanical forces. The second difference would be the lack of orientation control in the paper by Ossuma Khatib. As demonstrated in 5.4, orientation control does not come naturally with potential fields, but has to be modeled as an additional, artificial torque at the tool tip center point.

2.2 Comparison with conventional Trajectory Planning

Artificial potential fields were introduced to allow for real time obstacle avoidance. Even so Ossuma Khatib used a `PDP 11/45` for his experiments, the problem of conventional trajectory planning being to slow for real time applications still holds true today. Here is the potential field based approach clearly better suited, due to its simple core and its good scaling.

But potential fields also have some major drawbacks. The most notably being its limited abilities in terms of global path planning. The robot may easily get stuck in a configuration from witch it is unable to escape on its own or follow a non-optimal trajectory when moving from one point to another. Therefore the implementation presented here has been designed to be a subsystem, to be used in combination with a higher level intelligence for global path planning.

A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.

The Second Law of Robotics
by Isaac Asimov

3 Potential Fields

3.1 Mathematical Definition

In general, a potential field is a vector field which is the result of applying a gradient operator on a scalar field.

A scalar field is a space in which every point is assigned a scalar value. This can be denoted as a funtion f:

$$f : \mathbb{R}^n \to \mathbb{R} : \begin{pmatrix} x_1, & x_2, & \dots & x_n \end{pmatrix}^T \to y \tag{3.1}$$

where n is the spaces dimension.

The gradient operator is defined as:

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1}, & \frac{\partial f}{\partial x_2}, & \dots & \frac{\partial f}{\partial x_n} \end{pmatrix} \tag{3.2}$$

Thus the operator assignes every point a vector, pointing in the direction, that maximally increases f as seen from this point.[4]

3.2 Two Dimensional Potential Fields

For ease of understanding, two dimensional potential fields will be discussed first and will then be generalized for the three dimensional case.

The two dimensional scalar field can best be visualized by representing the scalar values as brightness values. This can be seen in figure 3.1. Of course the gradient

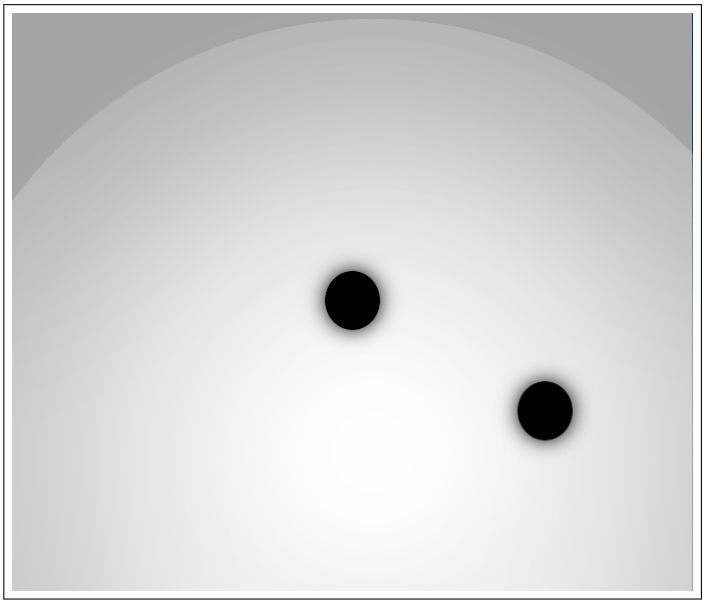


Figure 3.1: Two dimensional scalar field

operator can be applied here and results in the potential field in figure 3.2 For a more intuitive understanding the two dimensional scalar field can be represented as a plane in the three dimensional space. In figure 3.3 the XY-plane. Further the scalar values can be represented as deformations of the plane along the, until now unused, Z-axis instead of brightness values. See figure 3.4 (the overlaying wire frame has been added for better visibility only). For the intuitive understanding, in figure 3.5 is now added a gravitational field along the negative Z-axis and added a 'ball' as probe. As the ball rolls from higher to lower points, it follows the vector field. This can be illustrated by looking at the plane from 'straight above', thus transforming it back into a two dimensional field. For better contrasts in figure 3.6 the potential field is again shown as brightness values. The dashed line represents the path the ball follows.

During the steps explained above, the two dimensional scalar field has been used to generate an also two dimensional path from a start point to a goal point while avoiding certain areas.

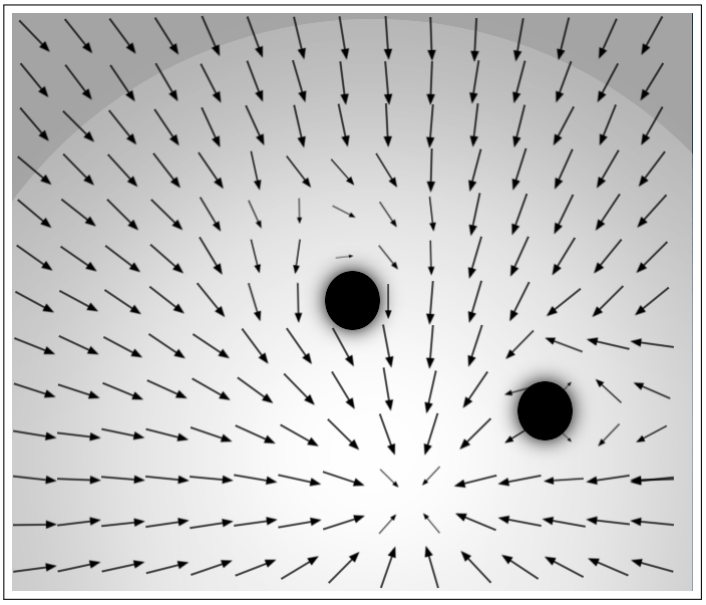


Figure 3.2: The resulting potential field

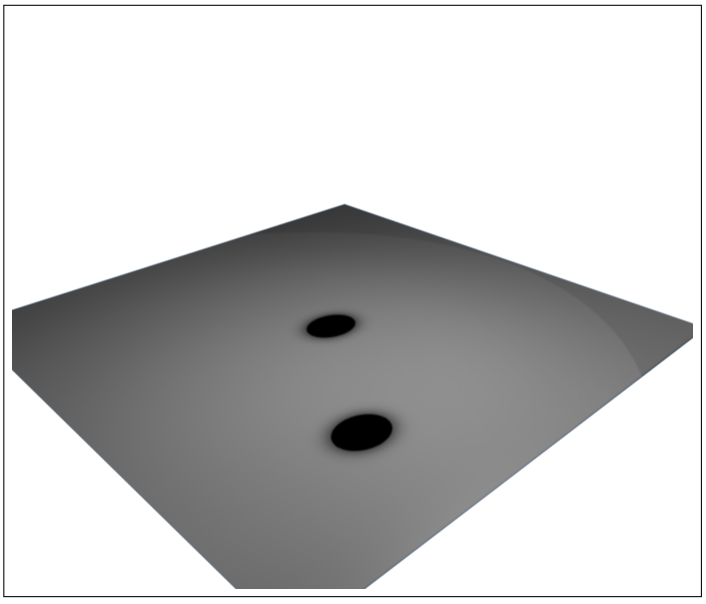


Figure 3.3: 2D plane in 3D space

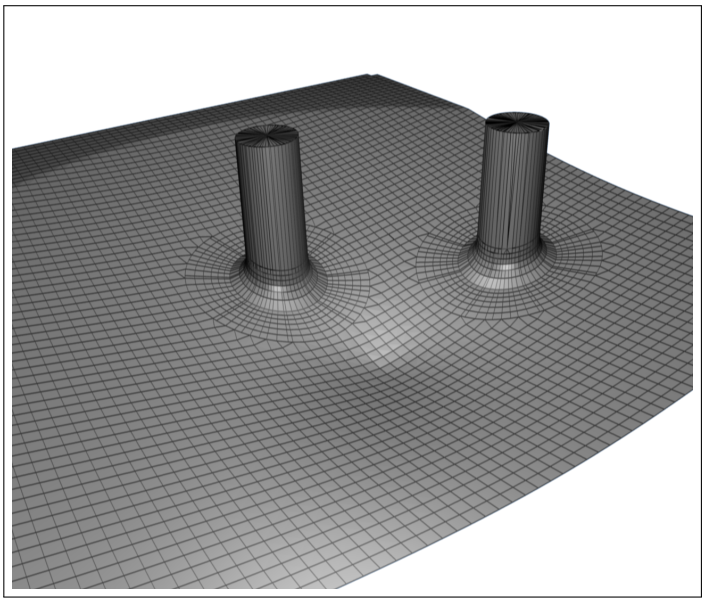


Figure 3.4: Potentials as deformation

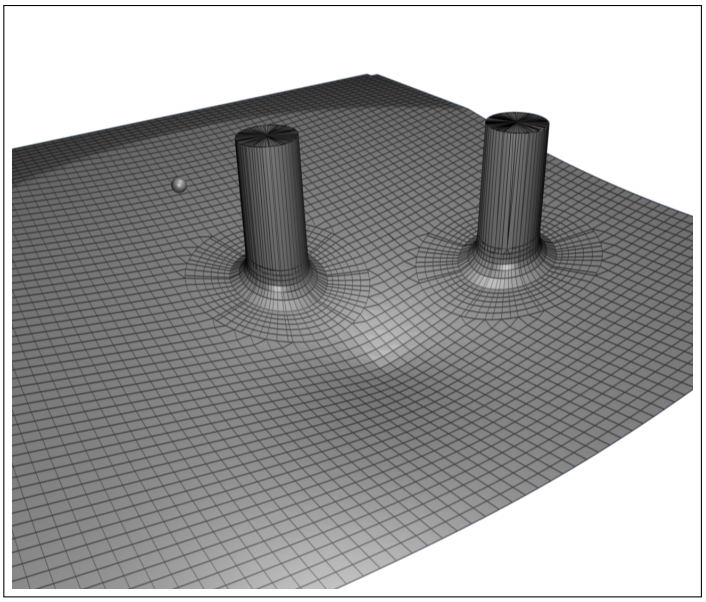


Figure 3.5: A probe is placed in the potential field

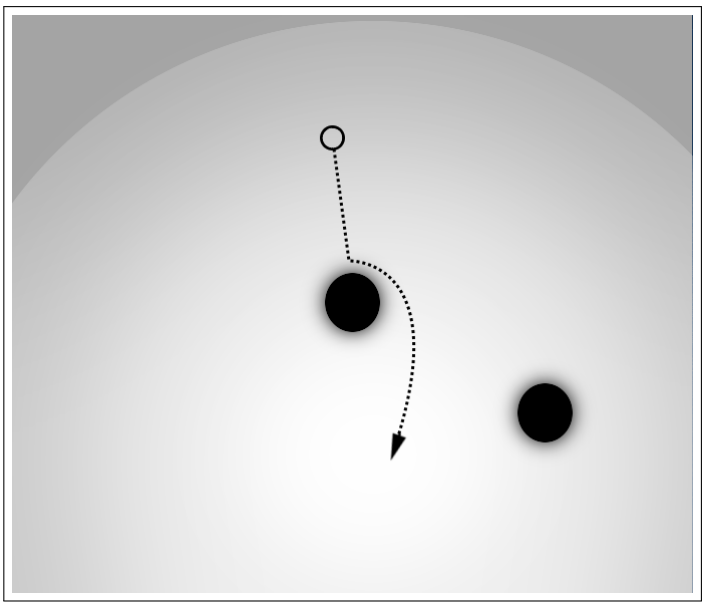


Figure 3.6: A probe moving in the potential field

3.3 The Potential Fields Design

The potential field is designed by applying two simple rules: Low values attract the probe and higher repel it. While multiple designs are possible only the one used later on will be described here in detail.

The potential field is composed of multiple independent functions, assigning a value to every point of the plane and merged by adding those values up. For the probe to move to a given point or to one of multiple given points these have to be the lowest values. Further the destination shall be reachable from every point, thus the gradient induced by the goal points has to spread the whole plane. For simplicity a linear gradient raising concentric from the goal point has been chosen.

$$v = d \tag{3.3}$$

Where v is the scalar value assigned to a given point and d is the distance of this point to the goal point.

In contrast, multiple obstacles may be added for the probe to avoid. For this explanation the obstacles are only single points, but it may easily be generalized to other shapes as well. The obstacle are composed of three parts, alligned as concentric circles. The inner part is a safety margin, never to be reached by the probe, therefore the value here is ∞[1]. The second part is the most important, where the probe is actually repelled. The last part of the obstacle is simply the area further away, where the obstacle does not have any more influence. The shown form was chosen for the following three reasons:

- Smooth transition from the outer to the repelling part
- The probe should be able to come close to the inner inner part
- But not to close. This means especially that it should never enter the inner part.

[1]For ∞ can not be represented it is shown as a flat plateau.

Dennoted as function:

$$v = \begin{cases} \infty & \text{for } i > d \\ \frac{1}{d-i} - \frac{1}{o-i} & \text{for } i > d > o \\ 0 & \text{for } d > o \end{cases} \tag{3.4}$$

Where i is the distance from the center to the inner boundary and o to the outer.

3.4 Critical Points

The major problem with potential fields is the possible disappearance of the gradient. As long as the gradients derivative, the *Hessian* matrix is non singular this may only happen at single, isolated points. The *Hessian* for the potential field illustrated here is not free of singularities. But since they may only appear in the inner part of obstacles, where they are unreachable for the probe, they can be safely ignored. For the remaining field three situations are possible in which the derivative may become zero:

The first two are the tips of maxima and saddle points. These two are extremely unstable, for the gradient is only zero in one singular point. Even the slightest movement will lead the probe to a point where the gradient is non zero and directed away from the trapping point. Experiments turned out, that errors, caused by sensor noise and numerical computation, are enough to ”free” the probe from such situations.

Local minima, in which the probe can get stuck before reaching the global minimum respectively the destination, behave differently. Unlike the two situation described before, local minima are stable. A slight movement will not free the probe, but direct it back to the trapping point. This is displayed in figure 3.7. Here the probe comes to a stop inside the U-shaped obstacle and is unable to reach the destination point from there. Different solutions to overcome this limitation exist, for example circulatory fields as proposed by Leena Singh et al. [11]. Circulatory fields not only repel the probe but also rotate it around the obstacle. Other popular solutions are evolutionary artificial potential fields as proposed by Prahlad Vadakkepat et al [12], or the Connectivity T^2 algorithm by Javier Antich et al. [2]. For evolutionary potential fields their parameters are variable and multiple configurations are simulated simultaneously, before they are compared by a special cost function, determining the best configuration, which is then used for navigation and as starting configuration for the modifed configurations during the next iterative step [12]. Connectivity T^2 algorithm is divided into two steps. The first one searches for all directions in which a movement is possible, the second one determines which of those is the most promising in order to reach the destination. For its decision a artificial potential field is used [2].

Since the system introduced here is designed only for simple navigation tasks or to be a subsystem for obstacle avoidance, guided by a higher level program, no such optimization has been included but would be possible, as referred to above, if more

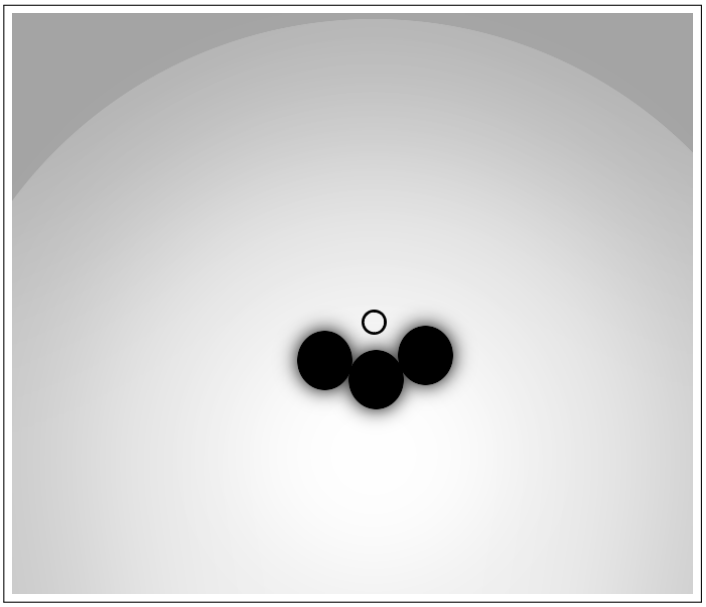


Figure 3.7: A probe trapped in a local minimum

sophisticated path planning is desired. When used in combination with a higher level system, the potential field allows for this to be slower than real time, only giving way points to the subsystem, between which the simple navigation abilities are sufficant.

3.5 The Three Dimensional Case

Until now only two dimensional potential fields have been discussed. The three dimensional case follows analogous, even though it is not as intuitive. Therefore the two dimensional case was discussed before.

The mathematical definitions given in the equations at the beginning of this chapter apply for all dimensions - in this case $n = 3$. Just as mentioned before every point can be assigned a scalar value and the gradient operator can be applied. This results in a three dimensional space in which every point is assigned a three dimensional vector, pointing in the direction locally maximizing the scalar value.

Everything else discussed here applies to the three dimensional case as well.

3.6 Numerical Consideration

Even so the generation of a potential field seems to be numerically expensive it may easily be drastically improved. The only interesting value is the effective vector at the probes position. Therefore only the effects of all obstacles and targets on the probe are of interest. All other points of the potential field can be ignored. Every one of those objects has a function for the effect they have on the probe based on there distance to the probe, as described before. The sum of all difference vectors corrected by the objects function equal the total force acting on the probe. This corresponds to the following equation:

$$x = \underbrace{\sum_{j=1}^{m} (g_j(|t_j - v|) \cdot (t_i - v))}_{targets} \underbrace{- \sum_{i=1}^{n} (f_i(|o_i - v|) \cdot (o_i - v))}_{obstacles} \tag{3.5}$$

Where x is the resulting vector, v is the position of the probe, n the number of obstacles, o_i is the position of an obstacle i, f_i is its function, returning a scalar scaling factor based on the distance between probe and obstacle, m is the number of targets, t_j is the position of target j and g_j is its function.

Therefore the complexity class is $o(n + m)$. Since the effect of every obstacle and target on the probe has to be considered this is optimal. A harmless complexity class is required, for the environment is potentially very complex, may change at any instance and the equation 3.5 has to be reevaluated at every incremental time step[2] to take this into account.

[2]This discussed in more detail in 4.1. For the setup given here one time step is 7.1ms.

A robot must protect its own existence
as long as such protection does not
conflict with the First or Second Laws.

The Third Law of Robotics
by Isaac Asimov

4 The Environment

4.1 The Mitsubishi RV-S6 Industrial Robot

4.1

The Manipulator

The RV-6S is one of the latest generation of six degrees of freedom articulated-arm industrial robots from Mitsubishi Electronic. All six joints are rotatory, stretching to maximum reach of 696mm. The basic composite speed of the hand wrist adds up to about 9300mm/s with a repeatability of ±0.02mm. The robot itself weights 58kg and is able to lift a payload of about 6kg. It is illustrated in figure 4.1. The hand wrist is equipped with a standardized gripper flange (`ISO 9409-1-31,5`). Inside the robot are six spare wires and two pneumatic pipes installed by the manufacturer. The pneumatic pipes have been replaced by a 15 wire cable and a heavily shielded[1] `IEEE 1394` firewire cable. [10]

[1]Inside the robot the cables run very close to the heavy servo motors, which are supplied by a pulse width modulator, generating a strong interference.

Figure 4.1: The Mitsubishi RV-6S

The Controller

The robot is controlled by a Mitsubishi CR2B-574 control unit. Besides controlling the robot it is also constructed control different additional systems with a set of specialized and general purpose IO connections. The controllers primary interface is a R45TB teaching panel with a 6.5″ touch screen for programming and monitoring the robot even during operation. For the purpose of this installation the controller has been equipped with an optional `CRn-500 10Base-T RJ-45` ethernet interface. Therefore the controller provides a telnet server, that allows to access the full range of available commands over a standard TCP/IP network. For this setup the connection

is only used to set the controller up for a special realtime mode. In this mode the controller expects a new set of configuration variables every 7.1ms, otherwise an automatic emergency stop is performed. Configuration variables are a complex, optimized bitfield which is sent as one UDP packet. One data packet is composed of:

- Command
- Type of data to be send
- Type of data to be received
- Position data
- Type of IO data to be send
- Type of IO data to be received
- IO data bitmask
- IO data
- Command counter
- Communication counter

This allows a maximum of control over the robots behavior but also makes the high level programming capabilities unaccessible. Besides emergency stops caused by critical errors, real time input is not subject to any further control or modification. For example giving a joint configuration that is not reachable within one time step will cause the servo amplifier to overload, and the controller to perform an emergency stop to prevent the hardware from being damaged. But directing the robot to a crash with its base plate will not be prevented by the control system [8; 9].

The Gripper

The gripper is a custom made construction, displayed in figure 4.2. From right to left it shows first the robots hand flange. Attached to this is an adapter for the following force torque sensor. This again is connected to an L-shaped base plate for camera and the gripper itself[2].

[2]The force torque sensor and the camera are not used for this setup.

Figure 4.2: The custom made gripper construction

4.2 The Vision System

The Vision system is not part of this Thesis. For the purpose of the test setup a basic vision system has been installed. It is based on the `OpenTL` general purpose tracking library, developed at the Chair for Robotics and Embedded Systems. Further details about `OpenTL` are available at `http://www.opentl.org/`. It is connected to the library described here with via an `telnet`-based protocol as described in 5.5.

For the vision system the robot has been equipped with three cameras. One is installed on the gripper, as can be seen on Figure 4.2. The other two are installed on a cage surrounding the robot. One vertically, one horizontally.

A robot may not injure humanity, or, through inaction, allow humanity to come to harm.

The Zeroth Law of Robotics
by Isaac Asimov

5 The Control System

5.1 Exertion of Forces

The chainlike construction of the manipulator requires the consideration of how forces propagate from joint to joint. A force acting on one joint causes a force in all lower joints that has to be acted against when pushing or pulling in the joints free direction to support a static equilibrium. For each joint, from highest down to zero, the force and torque exerted on by the previous link is calculated using the following equations:

$$^{i}f_i = {}^{i}_{i+1}R \cdot {}^{i+1}f_{i+1} \tag{5.1}$$

$$^{i}n_i = {}^{i}_{i+1}R \cdot {}^{i+1}n_{i+1} + {}^{i}P_{i+1} \times {}^{i}f_i \tag{5.2}$$

Where i is the current joint, ${}^{a}f_b$ is the force acting in b written in terms of joint a, analog for torques n, ${}^{c}_{d}R$ is is the rotation matrix from joint c to joint d and ${}^{e}P_f$ is the transformation matrix from e to f in terms of joint e. With these equations it is possible to introduce an additional force and/or torque at every step.

Finally the force required by the joint to balance this torque and force can be calculated. For a rotary joint the calculation is:

$$\tau_i = {}^{i}n_i^T \cdot {}^{i}\hat{Z}_i \tag{5.3}$$

Where ${}^{i}\hat{Z}_i$ is the normalized Z-axis of joint i in terms of joint i. Analog the actuator force for prismatic joints is[1]:

$$\tau_i = {}^{i}f_i^T \cdot {}^{i}\hat{Z}_i \tag{5.4}$$

[5]

[1] Only mentioned here for completeness. The Mitsubishi RV-6S does not actually have any prismatic joints.

5.2 Movement

The virtual force induced upon the tool tip center point by the artificial potential field are calculated as described in chapter 3, more precisely in equation 3.5. The forces acting on the other joints are calculated analogous except for the targets do not have any effect on them, for the target position is where the tool tip should be. The rest of the construction only has to avoid crashing into obstacles. Using these forces in combination with the exertion of forces described above gives a set of virtual forces acting in each joint. Like the ball in in chapter 3 rolls down the hill, the robot moves by giving in to these forces. This means the joint configuration for the next time step is the result of adding a value based on the acting force to the current joint configuration. The calculation of this increment is discussed in 5.3. During the next time step the joint forces are recalculated based on the robots new configuration and the potentially changed envoirement.

This results in the robot moving along a trajectory in the direction of the destination point while avoiding obstacles.

5.3 Joint Speed, Acceleration and Limit Control

Based on the force acting on a joint a rotation increment has to be calculated. This is subject to the following four factors:

- Acting force
  The force acting in the joint gives the direction and the base speed of the movement. Ideally the robot would directly follow this value. This would result in a fast movement when far away from goal point and from obstacles, in a slower movement around obstacles and finally phases out smoothly towards the goal. Yet the robots mechanical construction requires further limitations.

- Maximum acceleration
  As described in more detail in 6.1, this had to be determined by experiment for each joint and is not known exactly. Therefore a rather simple, constant acceleration function has been chosen.

- Maximum speed
  The maximum speed limitation has also been determined by experiment as described in 6.1.

- Joint limits
  For a smooth transition and to avoid the use of brakes or the mechanical end stops[2], the maximum speed has been limited by the distance to the joint limit as can be seen in figure 5.1. The closer the joint is to one of its limits the slower it can move in this direction until no more movement in this direction is possible, thereby coming to a slow stop. Joint limits have not only been used to adjust for the joints mechanical construction but also to prevent the robot from being damaged by self collision. All but collisions of the tool tip with robots base can simply be avoided by choosing appropriate joint limits without significantly impairing its mobility.

[2]Both are not designed to be used during normal operation but for emergency situations only. A regular usage might damage the robot.

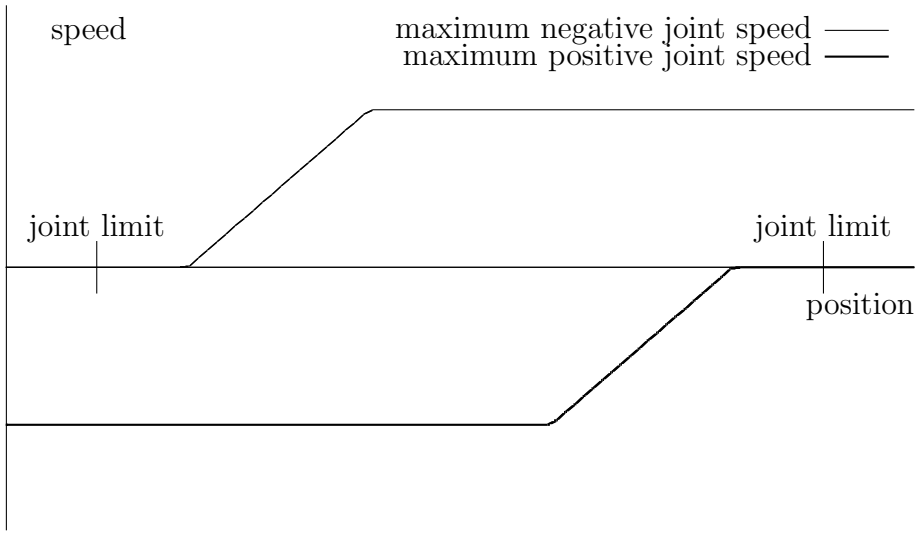


Figure 5.1: Joint speed limitation by proximity to absolut joint limit

5.4 Orientation

The tool tip orientation has not been discussed so far. In the classic potential field based approach this is not provided, yet it is required for most applications. To integrate the ability of choosing a desired orientation in the artificial potential field based system, forces acting on the robot, that direct the robot towards this orientation, are required. This is realized by adding an artificial torque acting on the tool tip.

The desired orientation as well as the tool tips current orientation can be described as a rotated standard coordinate system, defined by three three dimensional vectors. This is illustrated for the X-axis in figure 5.2. To translate the difference of those two coordinate systems in a torque the following equation is used:

$$n = \sum_{a=\{X,Y,Z\}} (a_d - a_c) \times (a_c) \tag{5.5}$$

a_d is the corresponding standard axis of the coordinate system of the desired orientation and a_c of the current.

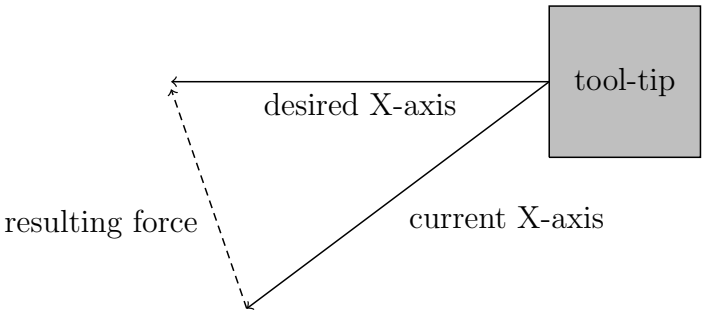


Figure 5.2: Torque causing reorientation

The use of a standardized coordinate systems results in a torque causing forces that clearly dominate over the values calculated by the potential field. Therefore a scalar factor adjusting the torque is introduced. This describes importance of reaching the correct position relative to reaching the correct position. An appropriate value was determined by experiment as described in 6.1.

5.5 Connection to Vision

For the test setup a connection to the vision system is required to receive information about the environments current configuration. This has been realized as a simple `telnet` like server and a custom protocol. The protocol definition can be found in appendix A.3.

Theory and practice sometimes clash.
And when that happens, theory loses.
Every single time.

Message to Linux kernel mailing list
by Linus Torvalds

6 Experiments

6.1 Parameter Determination

Maximum Joint Speed

The first parameter to determine by experiment was the maximum joint speed. Just by reading the absolute maximum speed for the robot given in the manual - like 400deg/s for the waist joint - it is obvious, that these values are way to large to be practical for the given task [10]. Including the gripper, the maximum tool tip center point speed sums up to more than 1m/s. Besides increasing the risk of destroying the robot during the development phase, it is also to risky during operation with a human operator near by. The robots mass of about 58kg and its powerful motors allow it to injour a human being seriously. Additionally the direction of a mass moving at this speed may not be altered fast enough as a result of the caused inertia and the position updates by the obstacle detection system might not be quick enough to avoid fast moving obstacles.

For the robots ability to change the grippers orientation or to avoid an obstacle with one joint, while maintaining a fix tool tip position, limiting the joint speed has been chosen over limiting the end effectors maximum speed.

Videos of all experiments are on the enclosed `CD-ROM`.

Maximum Joint Acceleration

While maximum joint speeds can be found in the robots official 'Standard Specification Manual'. The maximum acceleration is not specified, for the robots designed

way of programming using the `MELFA` programming language, the acceleration is automatically handled by the controller [9; 10].

The acceleration had not only to be limited for the robots capability but also for safety reasons as explained above. But yet another aspect had to be taken into consideration: fast speed changes might not be foreseeable for a human, decreasing his capability of avoiding a collision with the robot and impairing the experience of a natural movement.

Balance between Orientation and Position

The robot tries to achieve two goals, a position[1] and an orientation, which potentially exclude each other. The importance of the orientation relative to the position can be controlled by a scalar value. A value for the use in the test setup has been determined by experiment. Here avoiding obstacles is the center of attention, so the importance of the orientation could be set to a very low value.

[1] Moving towards the goal point and avoiding obstacles both control the position only.

6.2 Obstacle Avoidance Trial

The Test Environment

In the following, a few experiments conducted with the robot are presented to show both this techniques strengths and its weaknesses under realistic circumstances in a real application. For each experiment the setup is introduced, the observations made are noted and finally a conclusion is drawn from these observations.

For the experiments conducted here the robot has been mounted on a table. Around the robot a cage has been installed on which two cameras were mounted to observe the robots operating space. The robot is placed in one corner of the table and the robots shoulder joint movement is limited to a 90° angle in a way that the robot may only reach over the table, where an area of approximately 1×1m is observed by the cameras. The vision server is connected to control system, providing it with information about targets and obstacles.

Simple Movement

Setup:

The robots tool tip is initially positioned on one end of the observed area. An object, tagged as target by the vision system, is placed at the opposite site[2]. The tool tip orientation is neglected.

Observation:

The robots movement starts slowly but accelerates quickly. The tool tip moves towards the given destination. The trajectory does not follow a straight line but meander like line. The movement shows a very characteristic behavior: At first the joints close to the tool tip move considerably faster than those closer to the base, before those follow their movement. Finally the robot comes to a slow stop as the robot reaches the given target coordinates.

[2]For the robot would otherwise try to move 'into' the object an additional offset of 10cm is defined.

Conclusion:

Simple navigational task can be performed by the robot. The trajectory and the joint movement during the operation are suboptimal.

Simple Obstacle Avoidance

Setup:

The setup for this experiment is comparable to the one for the previous one, except this time an additional object is placed in the middle of the straight connection between start and destination point. This additional object is tagged as spherical obstacle by the vision system. The obstacle is positioned so no link will come close to it during movement.

Observation:

The robot moves towards the destination and the obstacle the same way it did during the previous experiment. As the tool tip closes in to the obstacle, it slows down considerably before performing a quarter circle movement around the obstacle. From this point it moves again as described before.

Conclusion:

Obstacle avoidance works well, yet the lack of global path planning is also observable. A movement along a slightly curved line around the obstacle would be preferable.

Advanced Obstacle Avoidance

Setup:

The robot is initially positioned in the middle of the previously defined area. The current position is given as destination point for the robot to maintain its position. An object, tagged as obstacle, then moved near the tool tip and the wrist joint.

Observation:

As the obstacle comes close to the tool tip, the robot leaves its position. When the obstacle is removed again, the robot returns to the destination point. As the obstacle moves towards the wrist joint, it evades the obstacle. The tool tip leaves the destination point but eventually returns, while the wrist joint still keeps distance to the obstacle.

Conclusion:

Obstacle avoidance has a higher priority than reaching the destination. Still the robot keeps trying to do so while evading the obstacle. Special attention should be paid to the robots ability to not only avoid a crash of the tool tip and the obstacle but all joints are able to avoid obstacles.

Movement in Combination with Orientation

Setup:

A pencil is placed on the table in the middle of the observed area. The robot starts from a distant location. For the robot to pick up the pencil not only the position is relevant but also the orientation. Position and orientation are delivered by the vision system. The following orientation as Roll-Pitch-Yaw Euler angles is aspired: $\left(0, \frac{7}{8}\pi, \phi\right)^T$ where ϕ is the orientation of the pencil lying flat on the table.

Observation:

The robot moves to the pencil as described above. In the beginning the given orientation is not maintained but again the tool tip moves faster in the direction of the destination than the base, resulting in an orientation completely different from the given one. As the tool tip closes in to the pencil, the orientation is progressively corrected. Finally the robot is able to grab the pencil.

Conclusion:

Orientation control works as desired. The scalar value to manage importance of the orientation relative to the importance of the position as described in 5.4 and

determined in 6.1 shows to be sufficient for this task, yet small enough to allow for a flexible movement. Other applications might have different requisitions and therefore require for this value to adapted accordingly.

Advanced Movement

Setup:

The robot shall again move from one end of the relevant area to the other. For this experiment an additional, vertical bar is added between the straight connection between start and end point and the robot. This experiment is conducted twice with two different start points, as illustrated in figure 6.1 and 6.1.

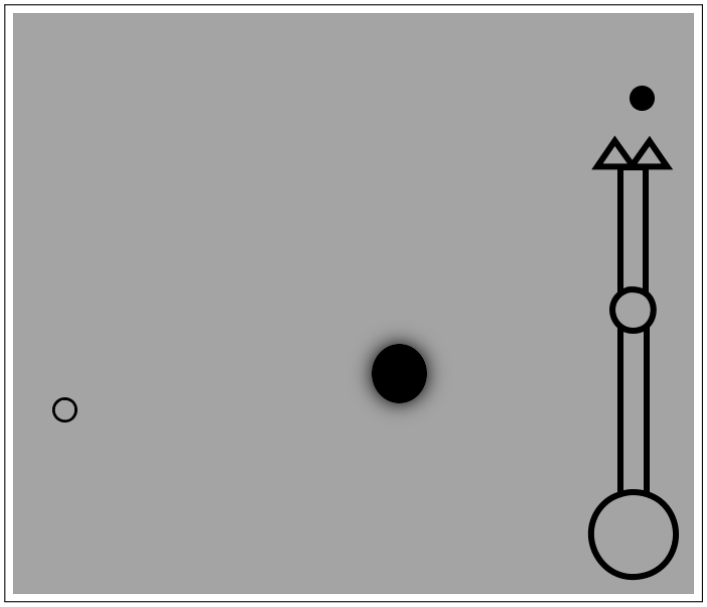


Figure 6.1: Setup for first advanced movement experiment

Here the operating space is shown from above. Start and end point are marked with small circles, the bar is marked as bigger circle. The shown robot is purely schematic.

Observation:

During the first experiment, the tool tip moves in the direction of the destination point. As the arm comes close to the bar, it stops before reaching the destination. It gets stuck in a position from which it is unable to proceed on its own. Only the tool tip stretches in the direction of the destination point, even when it is moved.

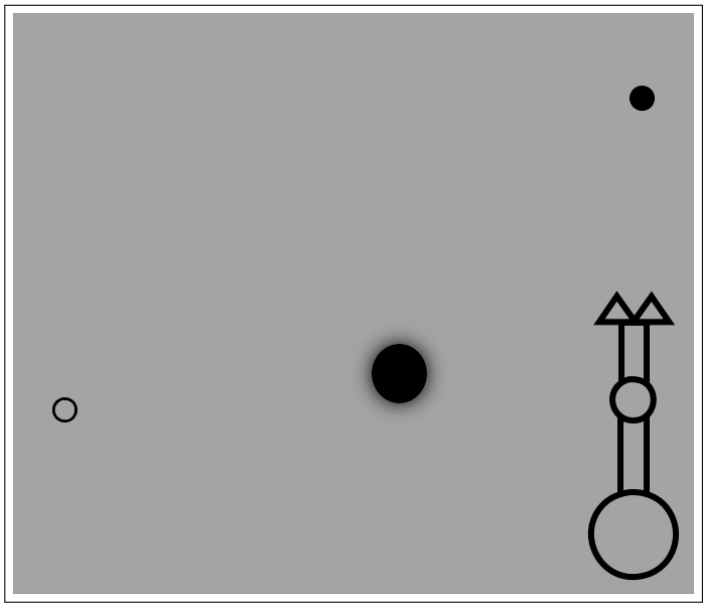


Figure 6.2: Setup for second advanced movement experiment

For the second experiment, the starting point is slightly moved. Some time is required for the robot to move around the bar in its way but eventually it is able to reach its destination.

Conclusion:

Global path planning is not possible without problems. While the robot may move around obstacles in a non-trivial way, reaching the destination can not be guaranteed, even when a path from start to end point exists.

On the other hand obstacle avoidance works well. The robot does not crash into the bar. Not only the tool tip but the whole robot stays close to the obstacle without hitting it.

Repeatability

Setup:

The robot is given two coordinates - $(0.3, -0.4, 0.5)^T$ and $(0.7, -0.2, 0.3)^T$ - to alternate between 20 times.

Observation:

	$\varnothing t$	$\varnothing \Delta x$	$\max \Delta x$
$\Delta\phi = 0.000704$rad/s $c = 0.001$	9.8s	1.826mm	2.972mm
$\Delta\phi = 0.000141$rad/s $c = 0.001$	15.6s	0.372mm	0.549mm
$\Delta\phi = 0.000141$rad/s $c = 0.002$	9.1s	0.235mm	0.377mm
$\Delta\phi = 0.000141$rad/s $c = 0.005$	5.2s	0.233mm	0.373mm
$\Delta\phi = 0.000014$rad/s $c = 0.005$	7.1s	0.028mm	0.050mm

The robot is considered stall and the mechanical breaks are activated, when no joint moves faster than $\Delta\phi$. c is a slow down factor to scale down from maximum joint speed[3]. See 6.1. $\varnothing t$ is the average time required to move from one coordinate to the other. $\varnothing \Delta x$ is the average and $\max \Delta x$ the maximum Cartesian position error when reaching the destination. As mentioned in 4.1, the robot guarantees a repeatability precision of ± 0.2mm during normal operation. Therefore an at least equivalent precision can be assumed for the values measured here.

The robot moves as described above. The path it follows is independent of speed and accuracy.

Conclusion:

Considering the robot has stopped moving, once all joint speed are below a certain threshold, could be shown to be an adequate approach. The resulting position is well within the limitations by physical construction. The robot moves very slow, when closing in to the destination. Especially for when a precision is required it takes a long time before the robot is considered stall.

[3]The maximum joint speed used here therefore equals the maximum possible joint speed times c

Science is what we understand well enough to explain to a computer. Art is everything else we do.

Foreword to 'A=B'
by Donald E. Knuth

7 Conclusion

7.1 Discussion

For this thesis, artificial potential fields and a control system based on them were introduced. This was not only done in theory but also implemented and on a real manipulator - a Mitsubishi RV-6S. It could be shown to be well suited for very fast responses to an environment that is completely or in parts unknown beforehand and may have a highly dynamic characteristic. The robots internal model of its surrounding is reevaluated constantly and very quick. Therefore the robot is able to react on changes in real time[1]. This does not only apply to the robots tool tip but stretches to all of it's joints allowing them to perform an obstacle avoidance of their own.

Of course this approach also has its draw backs. Most notably its inability to perform a global path planning. A good example for this problem is the experiment 'advanced movement' in 6.2. This constrain massively limits the possible operations without an additional higher level path planning algorithm. Such an algorithm may also be necessary to overcome an other limitation. Local minima[2] may cause the robot to get 'trapped' in a position, that is not the destination, from witch it is unable to move on on its own. An other draw back is its suboptimal joint movement. This also causes frequent changes in the speed mechanical components, especially for joints further away from the base. When used at higher speed this could cause problems due to the increased inertia.

[1] see 3.6

[2] see 3.4

7.2 Further Advances

Hardly any research has been done so far on artificial potential fields in combination with manipulator control. The most obvious first approach would be to back port advances, that have been made with potential fields in mobile robot navigation, to manipulators. A lot of which could be ported rather easily, some might require major changes, either for changing from the two dimensional case, typically used with mobile robots, to the three dimensional, required for manipulators, or for optimizations to take the fact into account, that manipulators are composed of multiple links, not just one as typical for mobile robots. But also further advances are possible requiring original research. For example, a smoother movement while avoiding obstacles with every link or detection of the final position. Currently the detection of the arrival at the final position is done by a minimal movement threshold for all joints.

This lack of prior research in the field of artificial potential fields for manipulator control can be seen as a rare opportunity for promising original, basic research.

7.3 Results

Since robot-human interaction is becoming more and more important for scientific research as well as for industrial applications, a reliable and fast system for obstacle avoidance becomes more and more important to allow for humans and robots to work in a joint operating space. In this thesis the approach for real time obstacle avoidance based on artificial potential fields for manipulator control was introduced and used to implement a control system. Further experiments could demonstrate its functionality in practice.

Even so research is still required in this field, artificial potential fields could be shown to be a viable approach to real time obstacle avoidance.

Bibliography

[1] http://sfb453.de/. Website.

[2] Antich, J., and Ortiz, A. Extending the potential fields approach to avoid trapping situations. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005)* (Aug. 2–6, 2005), pp. 1386–1391.

[3] Cahyadi, A. I., Ya-Chun, C., and Yamamoto, Y. Stable mobile robots teleoperation via potential field method. In *Proc. IEEE/ASME International Conference on Advanced Intelligent Mechatronics AIM 2008* (2–5 July 2008), pp. 347–352.

[4] Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L. E., and Thrun, S. *Principles of Robot Motion: Theory, Algorithms and Implementations*, 1 ed. The MIT Press, The MIT Press; 55 Hayward Street, Cambridge, Massachusetts MA 02142, USA, June 2005.

[5] Craig, J. J. *Introduction to Robotics Mechanics and Control*, 3 ed. Pearson Prentice Hall, Pearson Education, Inc.; Upper Saddle River, New Jersey NJ 07458, USA, 2005.

[6] Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. In *Proc. IEEE International Conference on Robotics and Automation* (Mar 1985), vol. 2, pp. 500–505.

[7] Mitsubishi Electric Coporation. *RV-6S Instruction Manual*, 1 ed., 2003.

[8] Mitsubishi Electric Coporation. *Bedienungsanleitung für CRn-500 Ethernet-Schnittstelle*, 1 ed., 2005.

[9] Mitsubishi Electric Coporation. *Bedienungs- und Programmieranleitung für Steuergeräte CR1/CR2/CR2A/CR2B/CR3*, 6 ed., 2006.

[10] Mitsubishi Electric Coporation. *RV-6S Standard Specifications Manual*, 5 ed., 2007.

[11] Singh, L., Stephanou, H., and Wen, J. Real-time robot motion control with circulatory fields. In *Proc. IEEE International Conference on Robotics and Automation* (Apr. 22–28, 1996), vol. 3, pp. 2737–2742.

[12] Vadakkepat, P., Tan, K. C., and Ming-Liang, W. Evolutionary artificial potential fields and their application in real time robot path planning. In *Proc. Congress on Evolutionary Computation* (July 16–19, 2000), vol. 1, pp. 256–263.

An algorithm must be seen to be believed.

The Art of Computer Programming
by Donald E. Knuth

Appendix

A.1 advancedRobot Manual

The full source code can be found on the enclosed `CD-ROM` along with `CMake` configuration files.

Requirements

- libRobot
  The libRobot library, developed at Chair of Robotics and embedded systems.
  An abstraction layer for connecting to different robot controllers.

- IPP
  Intel's Integrated Performance Primitives library.
  Available at `http://software.intel.com/en-us/intel-ipp/`

- Boost
  Extended C++ standard library.
  Available at `http://www.boost.org/`

- Boost Numeric Bindings
  Bindings for Boosts uBlas library.
  Available at `http://mathema.tician.de/software/boost-bindings`

Usage

The library provides four important classes. The central class, around which all other classes are designed, is the `advancedRobot`, representing the robot it self. It

only has to be provided with a `robotConfiguration`. A `robotConfiguration` is a simple structure, containing all of the robots characteristic data. For example joint limits. Such a one is provided for the Mitsubishi RV-6S.

To control the robot, it has to be connected to an `externalTargetHandler` and an `externalForceHanlder` using the functions `setExternalForceHandler` respectively `setExternalTargetHandler`. This two functions expect an point to a class derived from the corresponding handler.

The `externalTargetHandler` class has to be derived for usage. The only function, that has to be implemented is `getPositionsAndOrientation`. This function is not provided with any arguments and has to return a list of matrices. One 4×3 matrix per target, each representing a position in the first column and a coordinate system vector in each of the following columns.

The `externalForceHandler` class also has to be derived for usage and also only one function is mandatory: `get`. It is provided with a matrix $3 \times DOF$ (robots degrees of freedom). Each column representing the position of one of the robots joints. It has to return a matrix $4 \times DOF$ this time each column represents a force vector acting on the according joint as caused by some virtual entity like an artificial potential field.

For both, the `externalTargetHandler` and the `externalForceHandler`, a basic implementation is also given as `simpleTargetHandler` and `potentialField`. While the `simpleTargetHandler` only manages `targets`, the `externalForceHandler` can handle all kinds of obstacles derived from `object`. A few such `objects` are included. More precisely: `point`, `bar` and `plane`.

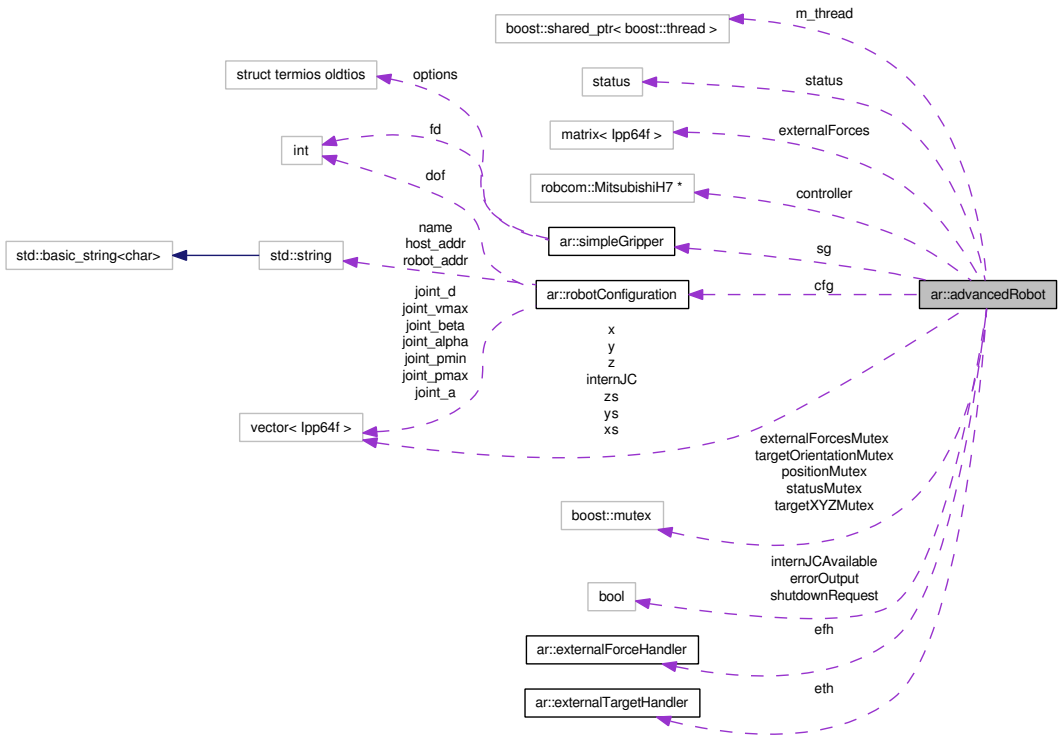
These implementation are again used by the `visionServer`. The `visionServer` provides a simple `telnet`-like sever over a standart TCP/IP network. Using the protocol defined in A.3 this server manges targets and obstacles. The functions `getExternalForceHandler` and `getExternalTargetHandler` return a pointer to the according handler to be connected to the robot.

A.2 advancedRobot API

In the following you find a very brief overview over the implemented library. It has been automatically generated using `doxygen`. A complete documentation along with the full source code can be found on the enclosed `CD-ROM`.

ar::advancedRobot Class Reference

A convenient abstraction of a real robot.
`#include <advancedRobot.hpp>`
Collaboration diagram for ar::advancedRobot:



Public Types

- enum **status** {
  STEADY, **MOVING**, **INITIALIZING**, **SHUTTINGDOWN**, **READY**, **ERROR** }

Public Member Functions

- ⋆ **advancedRobot** (**robotConfiguration** config)
- ⋆ ∼**advancedRobot** ()
- ⋆ rvector **currentPositionJC** ()
  currentPositionJC

- ⋆ rvector **currentTooltipPositionXYZ** ()
- ⋆ rvector **currentJointPositionXYZ** (int joint)
- ⋆ **status getStatus** ()
- ⋆ void **setExternalForceHandler** (**externalForceHandler** ∗x)
- ⋆ void **setExternalTargetHandler** (**externalTargetHandler** ∗x)

Public Attributes

- ⋆ **simpleGripper** ∗ **sg**

Detailed Description

This class allows the easy usage of a robot by utilizing a potential field based control system.

Member Enumeration Documentation

enum ar::advancedRobot::status

Enumerator:

STEADY
MOVING
INITIALIZING
SHUTTINGDOWN
READY
ERROR

Constructor & Destructor Documentation

advancedRobot::advancedRobot (robotConfiguration *config*)

Parameters:

config configuration to use to setup robot

advancedRobot::∼advancedRobot ()

Member Function Documentation

rvector ar::advancedRobot::currentJointPositionXYZ (int *joint*) `[inline]`

rvector advancedRobot::currentPositionJC ()

Returns:

current position as joint configuration

rvector ar::advancedRobot::currentTooltipPositionXYZ () `[inline]`

status ar::advancedRobot::getStatus () `[inline]`

void ar::advancedRobot::setExternalForceHandler (externalForceHandler * *x*) `[inline]`

void ar::advancedRobot::setExternalTargetHandler (externalTargetHandler * *x*) `[inline]`

Member Data Documentation

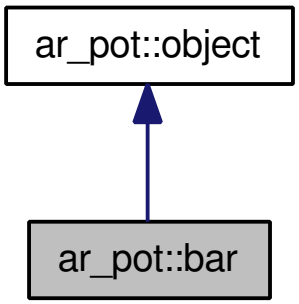
simpleGripper* ar::advancedRobot::sg

ar_pot::bar Class Reference

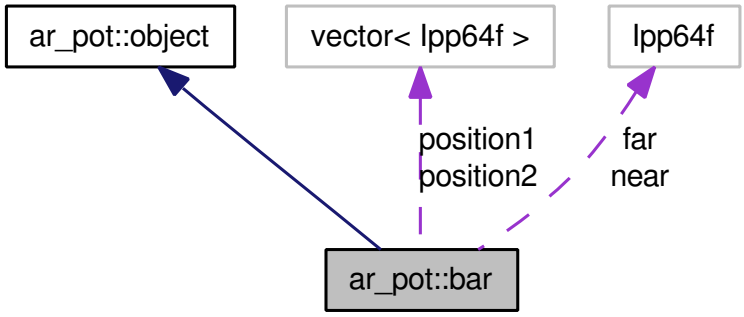
charged **bar** (p. 46) represents a repulsive bar-like charge in a potential field to represent bars as obstacles

`#include <advancedRobotPotentialField.hpp>`

Inheritance diagram for ar_pot::bar:



Collaboration diagram for ar_pot::bar:



Public Member Functions

- **bar** (rvector pos1, rvector pos2, Ipp64f near=0.1, Ipp64f far=0.2)
- ∼**bar** ()
- rvector **getPosition1** ()
- rvector **getPosition2** ()
- Ipp64f **inductedForce** (Ipp64f distance)

Constructor & Destructor Documentation

bar::bar (rvector *pos1*, rvector *pos2*, Ipp64f *near* = `0.1`, Ipp64f *far* = `0.2`)

Parameters:

pos1 position of one end of the **bar** (p. 46)

pos2 position of the other end

near radius for critical inner section

far radius for transisition zone

ar_pot::bar::∼bar ()

Member Function Documentation

rvector bar::getPosition1 ()

Returns:

position of one end **point** (p. 51)

rvector bar::getPosition2 ()

Returns:

position of one end **point** (p. 51)

Ipp64f bar::inductedForce (Ipp64f *distance*) `[virtual]`

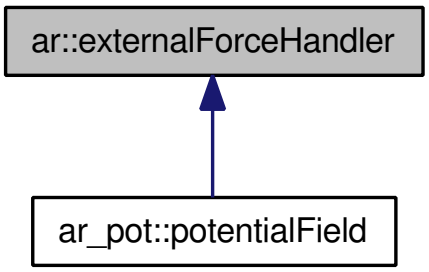
Returns:

scalar

Implements **ar_pot::object** (p. 49).

ar::externalForceHandler Class Reference

base class for an external force handler
`#include <advancedRobot.hpp>`
Inheritance diagram for ar::externalForceHandler:



Public Member Functions

- virtual matrix< Ipp64f > **get** (matrix< Ipp64f > positions)=0

Detailed Description

Derive this class to build your own external force handler and set it by calling **advancedRobot::setExternalForceHandler()** (p. 46). During each calculation cycle the function **get()** (p. 48) will be called.

Member Function Documentation

virtual matrix<Ipp64f> ar::externalForceHandler::get (matrix< Ipp64f > *positions*) `[pure virtual]`

Parameters:

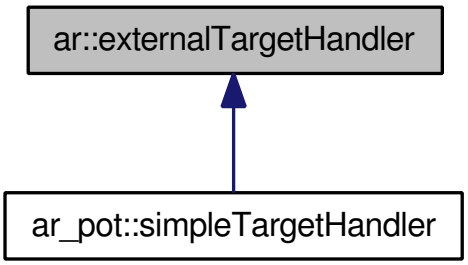
positions a matrix 3x(dof) with Cartesian positions of the robots joints in columns Override this function to return the forces acting on each joint in columns

Implemented in **ar_pot::potentialField** (p. 54).

ar::externalTargetHandler Class Reference

base class for an external target handler Derive this class to build your own external target handler and set it by calling **advancedRobot::setExternalTargetHandler()** (p. 46). During each calculation cycle the function **getPositionsAndOrientations()** (p. 49) is called.
`#include <advancedRobot.hpp>`

Inheritance diagram for ar::externalTargetHandler:



Public Member Functions

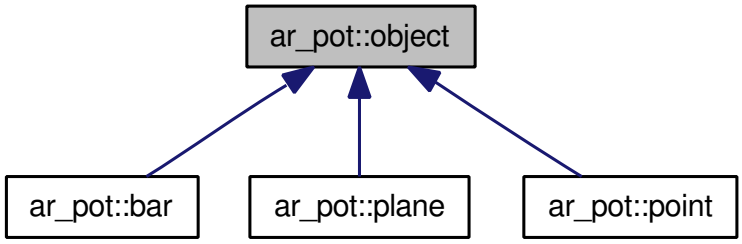
- virtual std::list< matrix< Ipp64f > > **getPositionsAndOrientations** ()=0

Detailed Description
Returns:

Member Function Documentation
virtual std::list< matrix<Ipp64f> > ar::externalTargetHandler::getPositionsAndOri
() `[pure virtual]`
Implemented in **ar_pot::simpleTargetHandler** (p. 58).

ar_pot::object Class Reference

base class for obstacles
`#include <advancedRobotPotentialField.hpp>`
Inheritance diagram for ar_pot::object:



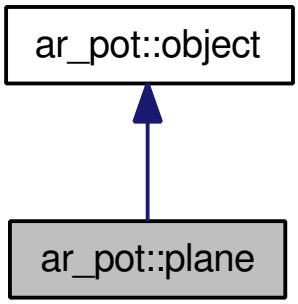
Public Member Functions

- virtual Ipp64f **inductedForce** (Ipp64f distance)=0

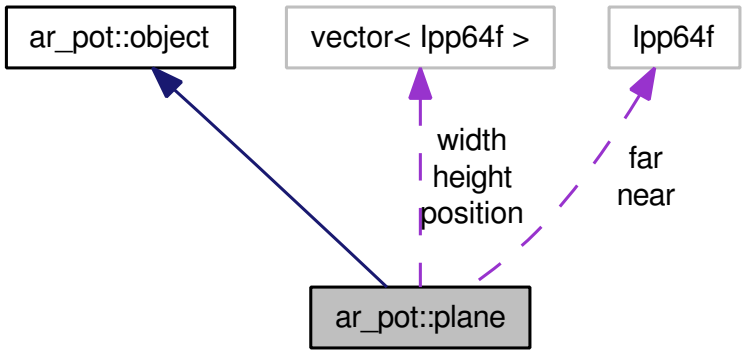
Member Function Documentation
virtual Ipp64f ar_pot::object::inductedForce (Ipp64f *distance*) `[pure virtual]`
Implemented in **ar_pot::point** (p. 53), **ar_pot::bar** (p. 47), and **ar_pot::plane** (p. 51).

ar_pot::plane Class Reference

charged **plane** (p. 50) represents a repulsive **plane** (p. 50) charge in a potential field to represent planes as obstacles
`#include <advancedRobotPotentialField.hpp>`
Inheritance diagram for ar_pot::plane:



Collaboration diagram for ar_pot::plane:



Public Member Functions

- **plane** (rvector pos, rvector width, rvector height, lpp64f near=0.1, lpp64f far=0.2)
- ~**plane** ()
- rvector **getPosition** ()
- rvector **getWidth** ()
- rvector **getHeight** ()
- lpp64f **inductedForce** (lpp64f distance)

Constructor & Destructor Documentation

plane::plane (rvector *pos*, rvector *width*, rvector *height*, lpp64f *near* = `0.1`, lpp64f *far* = `0.2`)

Parameters:

pos position of base (one corner)

width vector from pos to one corner

height vector from pos to one corner

near radius of critical inner section

far radius of transition zone

plane::∼plane ()

Member Function Documentation

rvector plane::getHeight ()

Returns:

height vector

rvector plane::getPosition ()

Returns:

position of base

rvector plane::getWidth ()

Returns:

width vector

Ipp64f plane::inductedForce (Ipp64f *distance*) `[virtual]`

Returns:

scalar

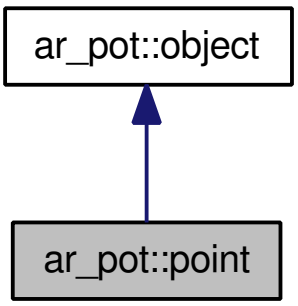
Implements **ar_pot::object** (p. 49).

ar_pot::point Class Reference

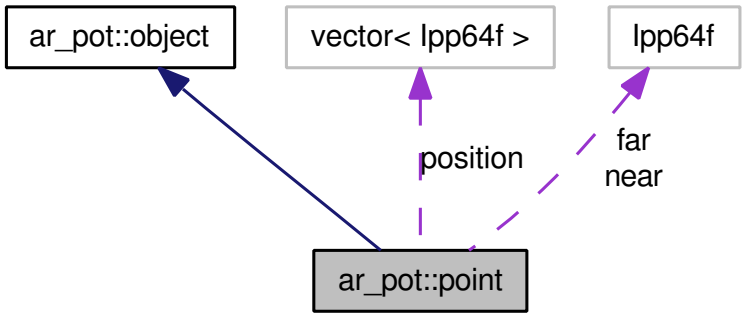
point (p. 51) charge represents a repulsive **point** (p. 51) charge in a potential field for obstacle modeling

`#include <advancedRobotPotentialField.hpp>`

Inheritance diagram for ar_pot::point:



Collaboration diagram for ar_pot::point:



Public Member Functions

- **point** (Ipp64f x, Ipp64f y, Ipp64f z, Ipp64f near=0.1, Ipp64f far=0.2)
- **point** (rvector pos, Ipp64f near=0.1, Ipp64f far=0.2)
- ~**point** ()
- rvector **getPosition** ()

  get position

- Ipp64f **inductedForce** (Ipp64f distance)
- void **setPosition** (rvector p)

Constructor & Destructor Documentation

point::point (Ipp64f *x*, Ipp64f *y*, Ipp64f *z*, Ipp64f *near* = `0.1`, Ipp64f *far* = `0.2`)

Parameters:

x position X

y position Y

z position Z

near radius of critical inner section

far radius of transition zone

point::point (rvector *pos*, Ipp64f *near* = `0.1`, Ipp64f *far* = `0.2`)

Parameters:

pos position as vector (X, Y, Z)

near radius of critical inner section

far radius of transition zone

point::~point ()

Member Function Documentation
rvector point::getPosition ()

Returns:

position of **point** (p. 51) center

Ipp64f point::inductedForce (Ipp64f *distance*) `[virtual]`

Returns:

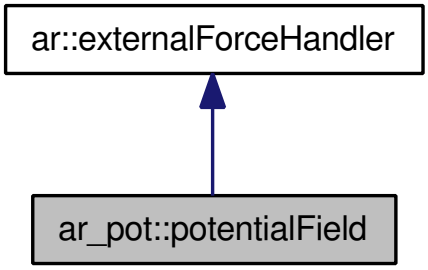
scalar

Implements **ar_pot::object** (p. 49).

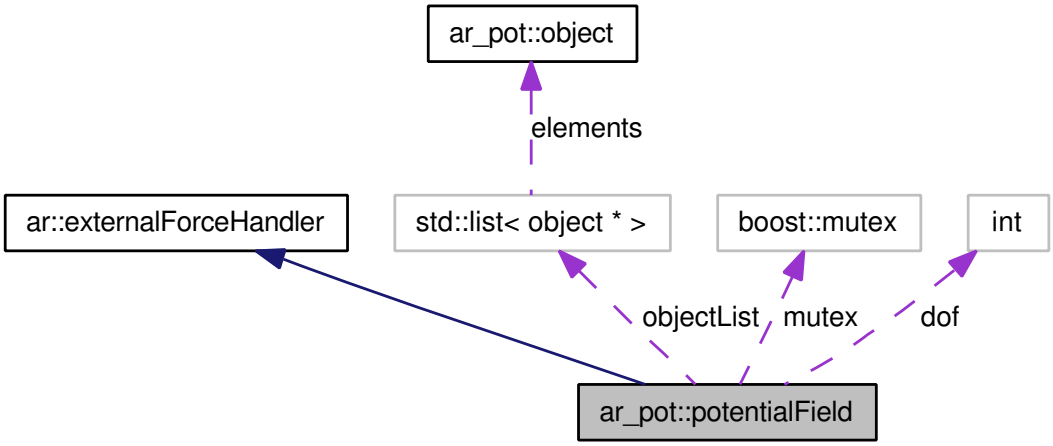
void ar_pot::point::setPosition (rvector *p*) `[inline]`

ar_pot::potentialField Class Reference

a potential field A potential field as an external force handler for ussage with advancedRobot::setExternalForceHandler().
`#include <advancedRobotPotentialField.hpp>`
Inheritance diagram for ar_pot::potentialField:



Collaboration diagram for ar_pot::potentialField:



Public Member Functions

- **potentialField** (int dof)
- ∼**potentialField** ()
- matrix< Ipp64f > **get** (matrix< Ipp64f > positions)

  get forces acting in a given set of points

- void **add** (**object** ∗obj)

  *add **object** (p. 49) to field*

- void **remove** (**object** ∗obj)

  *remove **object** (p. 49) from field*

Public Attributes

- boost::mutex **mutex**

Constructor & Destructor Documentation
potentialField::potentialField (int *dof*)

potentialField::∼potentialField ()

Member Function Documentation
void potentialField::add (object ∗ *obj*)

matrix< Ipp64f > potentialField::get (matrix< Ipp64f > *positions*) `[virtual]`

Parameters:

positions matrix 3xDOF with positions of each robot joint in columns

Returns:

matrix 3xDOF with forces acting on each joint in columns

Implements **ar::externalForceHandler** (p. 48).

void potentialField::remove (object ∗ *obj*)

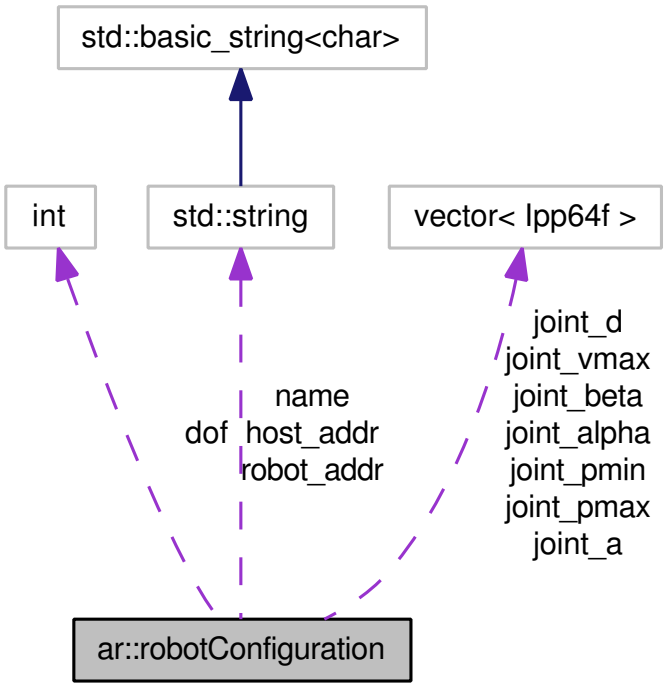
Member Data Documentation
boost::mutex ar_pot::potentialField::mutex

ar::robotConfiguration Class Reference

base class for robot configuration safes all kinds of values typicall for one specific robot

`#include <advancedRobot.hpp>`

Collaboration diagram for ar::robotConfiguration:



Public Attributes

- string **name**
- int **dof**
- string **robot_addr**
- string **host_addr**
- rvector **joint_a**
- rvector **joint_d**
- rvector **joint_alpha**
- rvector **joint_beta**
- rvector **joint_pmin**
- rvector **joint_pmax**
- rvector **joint_vmax**

Member Data Documentation

int ar::robotConfiguration::dof

string ar::robotConfiguration::host_addr

rvector ar::robotConfiguration::joint_a

rvector ar::robotConfiguration::joint_alpha

rvector ar::robotConfiguration::joint_beta

rvector ar::robotConfiguration::joint_d

rvector ar::robotConfiguration::joint_pmax

rvector ar::robotConfiguration::joint_pmin

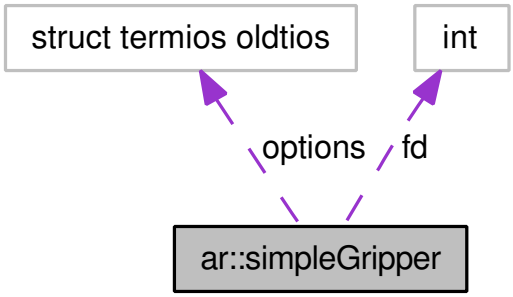
rvector ar::robotConfiguration::joint_vmax

string ar::robotConfiguration::name

string ar::robotConfiguration::robot_addr

ar::simpleGripper Class Reference

represents a very simple gripper Controls a gripper connected to serial port using the SV203 protocol
`#include <sv203.hpp>`
Collaboration diagram for ar::simpleGripper:



Public Member Functions

- **simpleGripper** (std::string dev)
- ~**simpleGripper** ()
- void **send** (std::string cmd)

  send command to device

- void **open** ()

  open the gripper

- void **close** ()

  close the gripper

Constructor & Destructor Documentation
simpleGripper::simpleGripper (std::string *dev*)

Parameters:

dev device identifier

simpleGripper::~simpleGripper ()

Member Function Documentation
void simpleGripper::close ()

void simpleGripper::open ()

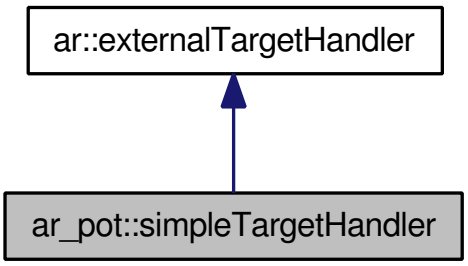
void simpleGripper::send (std::string *cmd*)

Parameters:

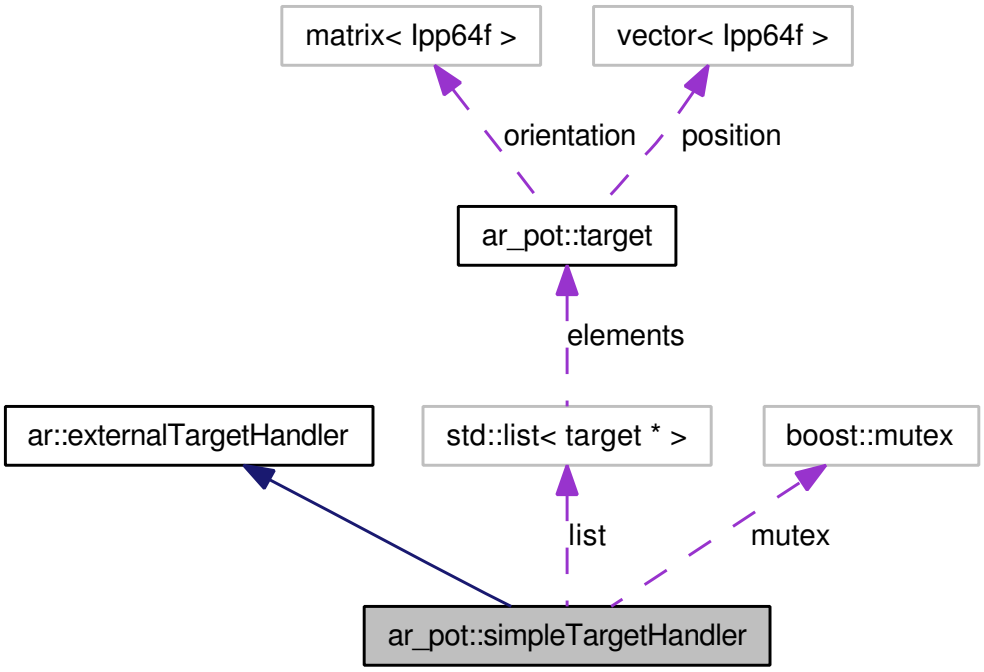
cmd command string to send

ar_pot::simpleTargetHandler Class Reference

potential field for targets represents a potential field for modeling multiple targets
`#include <advancedRobotPotentialField.hpp>`
Inheritance diagram for ar_pot::simpleTargetHandler:



Collaboration diagram for ar_pot::simpleTargetHandler:



Public Member Functions

- void **add** (**target** ∗t)
- void **remove** (**target** ∗t)
- std::list< matrix< lpp64f > > **getPositionsAndOrientations** ()

  get targets

Public Attributes

- boost::mutex **mutex**

Member Function Documentation

void ar_pot::simpleTargetHandler::add (target ∗ *t*) `[inline]`

std::list< matrix< lpp64f > > simpleTargetHandler::getPositionsAndOrientations () `[virtual]`

Returns:

a list of 4x3 matrixes each representing a position in first column and three orientation vectors in the following columns

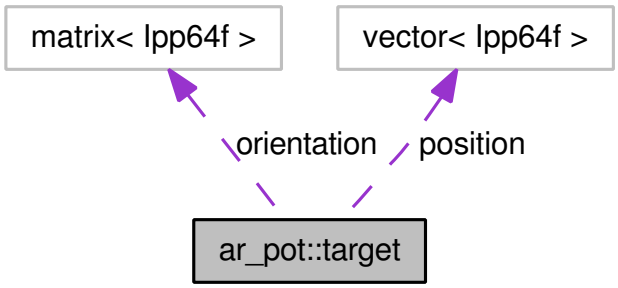
Implements **ar::externalTargetHandler** (p. 49).

void ar_pot::simpleTargetHandler::remove (target ∗ *t*) `[inline]`

Member Data Documentation
boost::mutex ar_pot::simpleTargetHandler::mutex

ar_pot::target Class Reference

target (p. 59) represents a **target** (p. 59) **point** (p. 51) and orientation
`#include <advancedRobotPotentialField.hpp>`
Collaboration diagram for ar_pot::target:



Public Member Functions

- **target** (rvector pos, matrix< Ipp64f > orient=zero_matrix< Ipp64f >(3, 3))
- rvector **getPosition** ()
- matrix< Ipp64f > **getOrientation** ()
- void **setPosition** (rvector pos)
- void **setOrientation** (matrix< Ipp64f > o)

Constructor & Destructor Documentation
ar_pot::target::target (rvector *pos*, matrix< Ipp64f > *orient* = zero_matrix<Ipp64f>(3,3)) [inline]

Member Function Documentation
matrix<Ipp64f> ar_pot::target::getOrientation () [inline]

rvector ar_pot::target::getPosition () [inline]

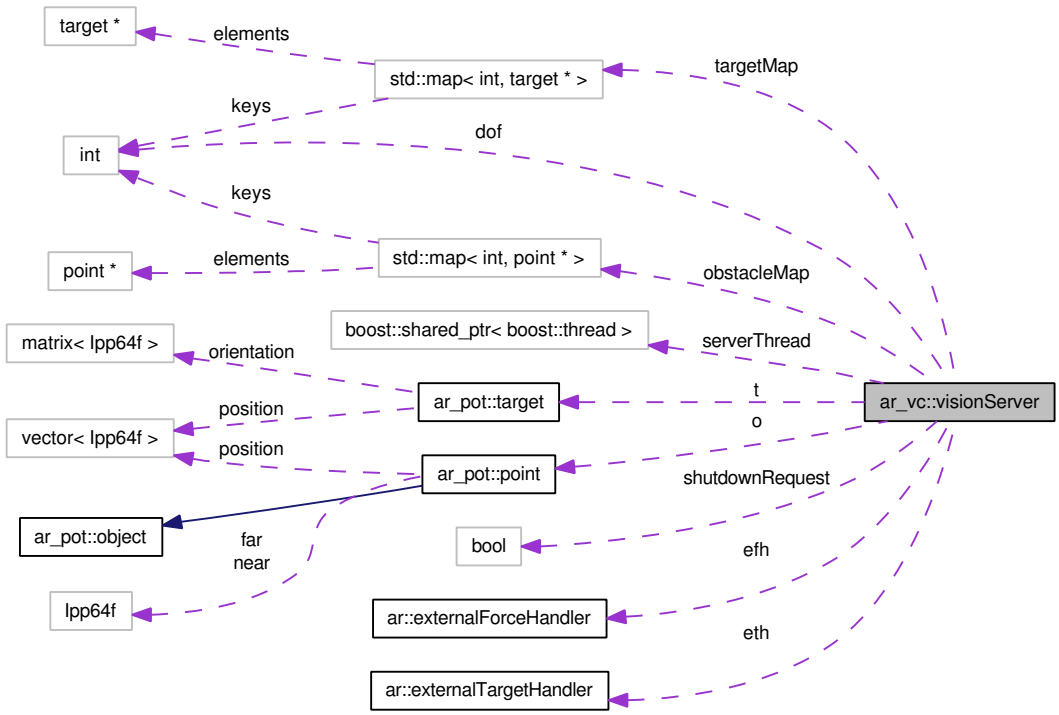
void ar_pot::target::setOrientation (matrix< Ipp64f > *o*) [inline]

void ar_pot::target::setPosition (rvector *pos*) [inline]

ar_vc::visionServer Class Reference

`#include <advancedRobotVisionConnector.hpp>`

Collaboration diagram for ar_vc::visionServer:



Public Member Functions

- **visionServer** (int dof)
- void **addStasticObject** (**object** ∗o)
- void **removeStasticObject** (**object** ∗o)
- **externalForceHandler** ∗ **getExternalForceHandler** ()
- **externalTargetHandler** ∗ **getExternalTargetHandler** ()

Constructor & Destructor Documentation
visionServer::visionServer (int *dof*)

Member Function Documentation
void visionServer::addStasticObject (object ∗ *o*)

externalForceHandler∗ ar_vc::visionServer::getExternalForceHandler () `[inline]`

externalTargetHandler∗ ar_vc::visionServer::getExternalTargetHandler () `[inline]`

void visionServer::removeStasticObject (object ∗ *o*)

A.3 Network Protocol Definition

The network protocol uses a simple `telnet` connection. Each commend is composed of six or seven elements. The elements are delimited by spaces, the lines are delimited by a new line character ('\n' in C/C++);

OPERATION TYPE ID X Y Z [PHI]

- `OPERATION (char)`
  requested operation
  - ⋆ 'A': Add
  - ⋆ 'U': Update
  - ⋆ 'D': Delete
- `TYPE (char)`
  element type
  - ⋆ 'T': Target
  - ⋆ 'O': Obstacle
- `ID (int)`
  unique identifier
- `X, Y, Z (float)`
  position (measured from robot base)
- `PHI (float - optional)`
  orientation (only for targets that lie flat on the base plate and shall be picked up)