UNIVERSITEIT VAN AMSTERDAM

BACHELOR OPLEIDING KUNSTMATIGE INTELLIGENTIE

Picking a flower with a KUKA youBot

Using A pathplanning in a 4 DoF configuration-space to evade surrounding obstacles*

Bachelor thesis
Credits: 18EC

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Author:
Ysbrand M. A. A. GALAMA
10262067

Supervisor:
Dr. Arnoud VISSER

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

27th June 2014

Contents

Abstract

This project conducts research on the potential of using the configuration space of a KUKA youBot to pick a specific flower. The goal of the project is to make an autonomous robot that is able to grasp a flower without damaging the environment. Applications that can use techniques to recognise and pick a flower exits mainly in the agricultural field, but other industries can also use similar autonomous robots.

The technique used in this project to calculate a path for the arm to the goal while evading the obstacles is a configuration space in the dimensions of the Degrees of Freedom of the robotic arm. In this project the configuration space of a youBot is build where a three dimensional point in world space can be reached. There also exists the potential of mapping three dimensional coordinates from a perceived point cloud as obstacles, which the robot should evade. Path planning in this configuration space makes it possible for the robot to evade the surroundings while grasping the correct flower.

Perception was planned to consist out of a point cloud from three dimensional image data, from which the flower could be recognised to get the coordinates of the goal. From the image data could also be extracted the obstacles that have to be evaded. However, due to limited resources the perception of the robot could not be finished. The movement planning of the system works promising and results in a well working kinematics for the youBot.

1 Introduction

There is an upcoming market for autonomous robots in industrial applications. Nowadays much work in the industry is carried out by robots, but most of them are not yet autonomous. Most robotic applications are either controlled by humans, or perform the exact same task repeatedly. Therefore research is conducted to develop more autonomous robots. One of the industries that have the need for autonomous robotics is the agricultural field. In this specific industry the challenge lies in not damaging the vegetation. As Siciliano and Khatib (2008, Chapter 46.) describe there are already many automated applications in the agricultural field, but improvement is needed. Fully autonomous agents that can execute their task without damaging itself or the surroundings do not exist in the field yet. Utilisation of such autonomous robots can majorly improve the farming on a large scale. When a robot is to be deployed in the field, a great reduction of work can be achieved with respect to human workers.

To facilitate research the robotics team from the University of Amsterdam (UvA) participates in the RoboCup@Work League. In this league one of the challenges is for an agent to navigate through a field of flowers and return with a specific flower. The team from the UvA does not build its own robots, but uses standard platforms build by others. The robot chosen for this task is the KUKA youBot (Bischoff et al., 2011). This robot is among others an educational platform on which researchers and students can test their algorithms.

Figure 1: A KUKA youBot on four omnidirectional wheels with a single 5 DoF arm

The youBot is a drivable platform on four omnidirectional wheels, by which it can move in two dimensions and turn on every point in 2D space, and a robotic arm of 5 degrees of Freedom (DoF). There are youBots with two arms as well, but the UvA robotics teams only uses a robot with one

arm. In figure 1 an image of the youBot can be seen.

Research by van Enk (2013) shows that navigation through a flowerbed is possible with A^* path planning. After a path is planned, the youBot can drive on its omnidirectional wheels through the field of roses to a goal while evading the flowers. Therefore this project focuses on perceiving and picking the flower with its arm while the assumption is made that the robot is positioned in front of the flower and is not required to drive to its goal. This results in a robot that stays stationary and does not use its wheels while using its arm to a position calculated form image data of a camera. In previous research of the UvA robotics team conducted by van Enk (2013) and Negrijn et al. (2014) the ROS interface for programming the robot was used. Therefore this project also makes use of this interface to maintain continuity and make distribution possible. If this program can make the youBot recognise and pick a flower, it could make a contribution to the RoboCup@Work team of the UvA.

The goal of this project is to be able to recognise and pick a given flower without damaging the environment. This project will focus on perceiving the surroundings and connecting them to a configuration space (Dorst et al., 1991) of the DoF of the youBot. In section 2 a detailed description on how this is achieved can be read. The `MoveIt!`(Sucan and Chitta) environment was deliberately not chosen due to the fact that it is not fully supported on the youBot interface. To make a flower-picking robot working, a standalone ROS plugin was made which should be able to achieve this task. The outcome of this plugin can be found in section 3, while in section 5 potential further research is stated.

2 Method

The problem that is undertaken in this project can be separated in two problems, firstly the perception of the environment and the recognition of the flower and secondly the movement of the robotic arm and the evading of the surroundings. For the second problem, as stated in the introduction, the choice was made to use a configuration space for the motion planning of the robot. To achieve this, several pieces of code had to be build.

Firstly, because the ROS interface works in `C++`, a multidimensional boolean array had to be build, with a conversion between the real world axis of the robot and the dimensions of the configuration space. More detail how this was achieved can be read in section 2.1. The array is boolean because some states cannot be reached; the states where the robot collides with itself or the environment. The fashion in which this is done can be read in section 2.3.

Secondly, after building the configuration space, a path planning algorithm had to be implemented to search for the best path, which is shown

in section 2.2. When a path is found, the joint states are send to the robot so the arm moves to the correct goal. The calculation between the world space and configuration space can be extracted from the kinematics of the robot, of which an image with the DoF is shown in figure 2

To be able to recognise a flower from the data of a camera, several options with a point cloud form a depth-camera can be set out. `C++` offers a library where point clouds can be analysed (Rusu and Cousins, 2011) which makes the recognition of a flower possible. However, due to the limited resources of this project, it was not possible to build such a plugin. More on possible vision of the robot can also be read in section 2.4 where the perception of the obstacles of the surroundings is laid out.

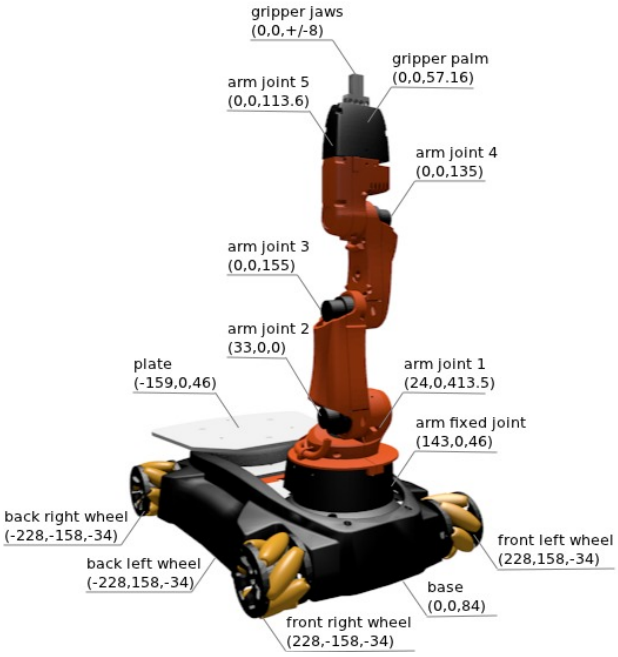

Figure 2: A model of the KUKA youBot with the positions of the joints (KUKA youBot Developers, 2011)

2.1 Configuraton space

As shown by Dorst et al. (1991) the configuration space makes it possible to search in the joint configuration for the shortest path without collision. The configuration space consists of a multidimensional space where each axis corresponds to one of the DoF of the robot. Therefore each point in this space represents a specific rotation in each axis and thus a single configuration of the robot. The youBot has a 5 DoF arm, but to reduce the complexity the rotation of the last joint has been rejected, thus a 4 DoF

configuration space is used.

For the fact that in a continuous configuration space a path cannot be calculated, the space is discretized. To reduce the complexity the precision of the discretising is 6^o or $\frac{\pi}{30}\,rad$ in all the remaining DoF. The resulting configuration space is therefore an array of dimensions: $56 \times 49 \times 26 \times 33$.

In the resulting space a path planning algorithm can search for the best path.

2.2 Path planning

The path planning algorithm that has been implemented is A^* (Hart et al., 1968). This algorithm, that is also used in two dimensions in the research of van Enk (2013), has been implemented in `C++` for the purpose of path planning in four dimensions.

The heuristic function that is used to estimate the remainder to the goal is calculated with the Manhattan Distance Function and the step cost is set to 1. To reduce the time complexity the estimation is calculated once and stored with the appropriate path. To find the best path, an ordered data structure of the estimations is used. In this structure the first element is always the path with the lowest cost and insertion is done in time complexity of $O(\log(n))$. However, the calculations to check if a node has already been visited is in $O(n)$.

At every iteration in the algorithm, a check is done at the current node to verify there is no collision at the next node. This collision could be a state where the arm intersects with a part of the robot, or an obstacle is on that location.

The goal that has to be found is the location of the flower, which is a coordinate in 3D world space. This coordinate can be deduced from the kinematics of the robotic arm. These kinematics can be calculated from the following matrix multiplication with rotations in 2D homogeneous coordinates, where (x, y, z) is the coordinate relative to the first joint and c_n and s_n are the $\cos(\theta_n)$ and $\sin(\theta_n)$ respectively:

$$\begin{pmatrix} \sqrt{x^2+y^2} \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} c_2 & s_2 & 33 \\ -s_2 & c_2 & 70 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} c_3 & s_4 & 0 \\ -s_3 & c_3 & 155 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} c_4 & s_3 & 0 \\ -s_4 & c_4 & 135 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 185 \\ 1 \end{pmatrix}$$

From this multiplication the following formulae are derived:

$$\tan(\theta_1) = \frac{x}{y}$$

$$\begin{aligned}\sqrt{x^2+y^2} =& 185 \cdot \big(s_4 \cdot (c_2 \cdot c_3 - s_2 \cdot s_3) + c_4 \cdot (c_2 \cdot s_3 + s_2 \cdot c_3)\big) \\ &+ 135 \cdot (c_2 \cdot s_3 + s_2 \cdot c_3) \\ &+ 155 \cdot s_2 + 33\end{aligned}$$

$$\begin{aligned}z =& 186 \cdot \big(- s_4 \cdot (s_2 \cdot c_3 + c_2 \cdot s_3) + c_4 \cdot (c_2 \cdot c_3 - s_2 \cdot s_3)\big) \\ &+ 135 \cdot (c_2 \cdot c_3 - s_2 \cdot s_3) \\ &+ 155 \cdot c_2 + 70\end{aligned}$$

Shown in figure 3 is a drawing that explains the kinematics of the formulae. With these formulae the states of the arm where the tip is at a goal point, the position of the flower for example, can be found. From there the arm can move to one of the possible end-states to grasp the flower.

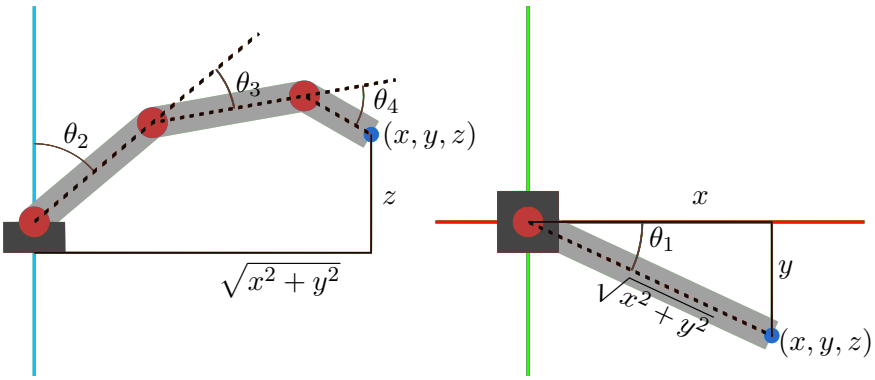

Figure 3: The kinematics of the KUKA youBot after eliminating the last DoF

Because the goal has several states in the configuration space, while the start is a single node, the choice was made, with a hint from Leo Dorst, to calculate the path from the goals to the start. Therefore the search for the final path becomes more trivial, because A^* is faster with a single goal, while beginning with several start nodes does not affect the performance. In the end the path is reversed so the robot can move with the correct steps to the goal.

2.3 self-collision

It is important to eliminate the nodes in the configuration space where self-collision is possible. There are several configurations for the robot where

the tip of the arm collides with another part of the robot, and if the robot tries to reach these configurations, damage can be caused to the hardware of the robot.

To detect self-collision a simulator has been built in Blender3D (Blender Foundation, 1995). In this simulator a simplified model of the youBot was made from the model of KUKA youBot Developers (2011) and a script was ran to detect the collision states. Usage of Blender3D was chosen for the previous experience with this program.

The program, made in Blender 2.49, rotates the arm according to the possible states in the configuration space. At each state there is calculated to check if the robot intersects with the fourth and final part of the arm the base and first two parts of the arm. It is not necessary to calculate for intersections with the other parts, because they cannot intersect in without intersection of the fourth part. To calculate if there is an intersection between the parts at each vertex of the fourth part a check is done if the vertex is inside the mesh of the other part. These calculations exist native in Blender, therefore only some small calculations where necessary to convert the points between frames of reference from local to global space. Whether an intersection, thus a self-collision, is found, the output is written to a file that can be read by the main program of the configuration space to include the self-colliding states of the robot in the path planning.

The calculating at which states the robot collides with itself takes a sizable amount of time, however, this program only has to run once, from which point on the self-collision states can be read in from a single file. The only time when this algorithm has to be ran again, is when the resolution of the program has to be more exact.

2.4 Obstacle detection

The KUKA youBot does not have sensors to detect the surroundings, therefore the RoboCup@Work team of the UvA makes use of a stereoscopic camera and couples this to the youBot. From the stereoscopic camera data in the form of a point cloud can be collected, these points in the point cloud are usually 3D coordinates. Several algorithms could be used to detect the flower from this point cloud and give the resulting coordinates to the path plan algorithm. However, due to limited time, this could not be implemented.

After the flower is found, the remaining point cloud data could be used to detect obstacles, which should be evaded with help of the configuration space search algorithm. The algorithm could plan around the obstacles if the point cloud coordinates are mapped in the configuration space. A possible manner to achieve this could be similar to the self-collision detection.

Therefore another simulation in Blender is scripted where an intersection of a single point with the robot is calculated. This program discretizes the

world space to voxels and at each state in the configuration space an iteration is done over the parts of the robot and saves the states where a voxel is intersected by a part. To speed up the calculation the base of the youBot and the first rotating part of the arm is only calculated once, because those voxels will always intersect with the robot, regardless of the configuration of the arm. From the database made with this program each point of the point cloud could fill the configuration space with collision states. However the time complexity of this program is too large to run with the limited resources that were used in this project, but another project could run and test this program. Though this program is only needed to run once as well. When a database of obstacle voxels is build, only the database has to be read to fill the configuration space.

3 Results

The plugin for the ROS interface that was built during this project has been tested with the components that were finished during the course of the project.

The A^* algorithm can find a path in the 4D configuration space with relative ease, as table 1 shows. In this table the size of a path and the initial states of the goal are shown with the time it took to calculate the path on a 64-Bit i5 processor. The found path can then be calculated back to the real world so the robot can move its arm to the given 3D coordinate.

Table 1: Several tests on the time performance of the A^* algorithm in the 4D configuration space

path size (# of nodes)	# of goal states	time (sec)
40	5	0.02
52	3	0.03
74	16	0.03
50	28	0.02
47	4	0.02
71	3	0.04

The self-collision states in the configuration space can be read from a file, made with the Blender script. The robot is tested with and without this file, and when tested without considering the self-collision states the robot arm tried to move through itself, when tested with the file, the robot moved without trying to go through itself. A short clip of the moving robot can be found in figure 4[1]. This film shows the youBot moving to several positions in the 3D world space.

[1]this clip does not work in all versions of pdf-readers

Figure 4: A short clip of the youBot navigating its arm to several coordinates in space.

Testing if the algorithm could evade obstacles could not be done, partly because the point cloud data could not be incorporated and partly because the building of the collision state took a considerable amount of time.

4 Conclusion

The usage of an A^* path planning algorithm in a 4D configuration space appears to be a elegant solution to move the robot arm in an obstacle filled space. Both time and space complexity of the program itself are surmountable, though the making of a filled configuration space considering the collision states are sizable. However, the making of a filled configuration space is needed only once, after which a database exists from which the program can read its collision states. The advantage of this system is that it is generalizable for other robots, with a small number of adaptions and a model of the robot, they could use this system.

The fact that the sensor data could not be processed due to the limited time, made it not possible to test the ability of the robot to pick a flower. However, in future research with this system, it could be possible to pick the correct flower while evading the surroundings. When the system is finished, the performance of the program can be evaluated on time complexity and accuracy.

5 Discussion

While the parts of the system that have been build work satisfactory, the goal of the project could not be reached. Due to limited resources it was not possible to finish the system and test it on the robot. While several frameworks were thought out to make the robot able to recognise and retrieve the flower, the programming of them took a large amount of time which made it impossible to build the complete system. With a smoother start in the

`C++` language or several weeks extra to program the system, the frameworks could have been finished what would make evaluating possible. Another possibility that could have resulted in a youBot that is able to pick a flower was a more extensive literature research on groups that are developing for the KUKA youBot presently. If more frameworks could have been used from other researchers, combining them in a working flower picking robot could have taken less time.

The framework build in this project has promising results, therefore further development of the algorithm is valid for research. The results of the finished parts of the framework were expected to appear as they are. The working A^* algorithm could find a path in the four dimensional configuration space because the A^* algorithm is known for its scalability in complex dimensions. The therefore working kinematics through the configuration space were even so expected.

The part of the project that could not be finished is the perception of the flower and obstacles and calculating these into the configuration space. It is possible to move to a given coordinate in 3D space, so the flower recogniser should return a coordinate where the robot needs to pick the flower. The evasion of the surroundings could also be implemented in a similar manner because with several adaptations the configuration space could be filled with the data of the collision script made in Blender.

Several steps could be undertaken to improve the current program. One of which could be a better heuristic in the A^* path plan algorithm, one which does not only take in account the distance in the configuration space, but moreover the distance in the 3D world space. The same modification can likewise be made in the path cost of the algorithm.

Another improvement could be a optimisation of the different parts of the program. There are several possibilities in the `C++` code for the calculation of the path as in the `Python` code within the Blender program to have less operations while executing the code. A small amount of optimisation has already taken place, but improvements from specialist could be made.

5.1 Further research

One possible future study could be finishing this project. Due to limited time, it was not possible to finish the program, therefore another project could be started which finishes the framework suggested in this thesis. The flower recognition could be done form the point cloud data, and the coordinates of the obstacles could be mapped in the configuration space. When this is done, the youBot should be able to pick the flower without damaging the environment.

Once a working program is build that can pick the flower with usage of the configuration space, several steps could be taken to optimise the program. The code that uses the A^* algorithm could possibly be more efficient

when optimised to the least complexity. The program that calculates the self-collisions and the program that calculates the obstacle collisions could additionally be optimised to be more accurate and less time consuming.

When the program is optimised, tests could be done to investigate if other path planning algorithms could find the goal state more easily. Another improvement could be the use of probability distributions to have a better understanding of the changing surroundings with respect to the theoretical calculations. Furthermore, work could be done to develop a `MoveIt!` package to make use of the libraries of `MoveIt!` within the applications of the KUKA youBot in agricultural environments.

References

R. Bischoff, U. Huggenberger, and E. Prassler. Kuka youbot-a mobile manipulator for research and education. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

Blender Foundation. Blender 3D, 1995. `http://www.blender.org/`.

L. Dorst, I. Mandhyan, and K. Trovato. The geometrical representation of path planning problems. *Robotics and Autonomous Systems*, 7(2):181–195, 1991.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

KUKA youBot Developers. KUKA youBot kinematics, dynamics and 3D model, 2011. `http://www.youbot-store.com/youbot-developers/software/simulation/kuka-youbot-kinematics-dynamics-and-3d-model`.

S. Negrijn, J. Haber, S. van Schaik, and A. Visser. Uva@ work customer agriculture order. 2014.

R. B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

B. Siciliano and O. Khatib. *Springer handbook of robotics*. Springer, 2008.

I. A. Sucan and S. Chitta. MoveIt! Online Available: `http://moveit.ros.org`.

J. van Enk. Navigating youbot through a rose field with A^*. *Project Report, Universiteit van Amsterdam. Science Park 904 1098 XH Amsterdam*, 2013.