International Journal of Advanced Robotic Systems

OPEN ACCESS
ARTICLE

Dynamic Arc Fitting Path Follower for Skid-steered Mobile Robots

Regular Paper

Peter Lepej[1]*, Johannes Maurer[2], Suzana Uran[1] and Gerald Steinbauer[1]

1 Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia
2 Institute for Software Technology, Graz University of Technology, Graz, Austria
*Corresponding author(s) E-mail: peter.lepej@gmail.com

Received 28 May 2014; Accepted 08 July 2015

DOI: 10.5772/61199

Abstract

Many applications, such as surveillance, inspection or search and rescue operations, can be performed with autonomous robots. Our aim is a control of modular autonomous systems in rescue robotics. One of the basic problems with autonomous robotics is the execution part where the control commands (translation and rotational velocities) are produced for mobile bases. Therefore we have focused on this area because there is only a small amount of available path following software for skid-steered mobile robots. Our goal was to develop a velocity controller that could be used for multiple skid-steered mobile bases. We considered differential drive mobile bases such as tracked skid-steering mobile bases. Our approach is based on an arc fitting algorithm, which takes into account the robot constraints and kinematical model. It produces a continuous trajectory where fitting to the given path is adapted based on given parameters. Moreover, we have included orientation angle compensation while the mobile robot is moving and ground inclination compensation. Our rescue robot is described, together with the simulation setup and algorithm implementation. We compared our algorithm to the Hector-based software and curvature velocity approach. The results for the proposed algorithm are shown for the simulation results and the experiment.

Keywords Mobile Robot, Arc Fitting, Path Following, Velocity Control, Rescue Robot

1. Introduction

Autonomous systems for controlling mobile robots in structured environments are complex and modular. In this work we focused on the sub-segment velocity control of an autonomous system, which is known under synonyms such as path following, path execution, tour control, motion control and pure pursuit, all of which are known to us. In this paper we will address our algorithm as the Dynamic Arc Fitting (DAF) path follower. We have used dynamic arc fitting to approximate the robot's local path using basic circular shapes. The circle's radius has been dynamically changed based on a given local path structure. Then we present our motivation and up-to-date know-how relating to this work. Our software for the autonomous rescue robot is based on software developed by the Hector Rescue Team from Darmstadt, Germany. The software is developed within the Robotic Operation System (ROS) [1]. The Hector software consists of Simultaneous Localization and Mapping (SLAM) [2] or the Hector_slam software package, which includes odometry-free mapping and a map and trajectory saver. Additionally, for autonomous navigation

Hector provides a Hector_navigation [3] stack for mobile robots. The navigation stack includes costmap and elevation mapping, an exploration planner and a controller. The Hector mapping and exploration stack has been proven at RoboCup competitions by the Hector Team (RoboCup 2013, Best in Class Autonomy award) and works very well and for this reason we chose it for our robot. In order to complete an autonomous system we needed a path follower, which has also been provided by the Hector stack. After some test runs we noticed that there were problems with the path follower, where the Hector path follower is not completely suitable for skid-steering robots and we were unable to successfully use the Hector path follower with our robot. This led us to a complete modular control system with a path follower for tracked skid-steering platforms. It was for these reasons that we developed our own path follower, which was tested and proven at a RoboCup competition.

General forms and references for path following and motion control are in [4], where car-like vehicles are considered. The described approaches can be applied on skid-steered mobile robots without minimum turning radius constraints. Work in [5] and [6] presents a smooth path construction when considering nonholonomic car-like drive. The control methods for steering mobile robots are presented, including collision checking. The work described in [7] focuses on smooth tour construction by predefining control path points. A kinematical model of a differential drive wheeled mobile robot was used during this experiment. The authors tried to optimize distance or time travelling to obtain the best possible tour for the mobile robot by considering minimal turning radius. Comparisons between various tour smoothing algorithms are presented as simulation results. In [8] the authors did not perform a practical experiment, but instead during simulation tried to fix the robot's position relative to the given circular path. Correction was done as a newly correct orientated circular path that intersects with the given circular path. The work done in [9] was executed in two steps, whereby the optimal connection with the robot's current position and offset trajectory was planned. During the second step the so-called move dependency kept the robot on the desired path. The authors present only the simulation results. The real problems with path execution are described in [10] and point out why their work would be suitable for high-speed robots. The authors provide an argument that ideal path following depends on a robot's current positional information. In [11] an algorithm is presented for generating paths with the help of Bezier curves, where so-called velocity patterns are used. And the path execution could be adjusted with parameters. The open-source hector_path_follower is within the Hector navigation package which is optimized for nonholonomic robots. Therefore the hector_path_follower is based on the pose_follower [12], which is integrated within ROS. The pose follower is based on simple positional and heading difference calculations. The translational velocity is calculated using simple transformation, being the multiplication of distance difference from the current and next waypoint and a transformation factor provided by a user. The resulting velocities are bounded by thresholds and rotating at place ability. The collision checker is used to check if the calculated path is collision-free, and if not, scaling down is performed over velocities. We have compared our approach with the pose follower and curvature velocity approaches, which are based on the Dynamic Window Approach (DWA). The curvature velocity approach has advantages over the classic path following methods because it includes collision checking. Collision checking is needed within dynamical environments, which also increases the complexity of the algorithm. The software is available in ROS under the move_base software stack. Its implementations are described in [13-14] and can be based on DWA or Trajectory Rollout (TR). Both approaches have sampling control space with available velocities where they differ based on how the robot control space is sampled. TR samples from the set of achievable velocities over all available environments, whereas DWA samples only in the set of achievable velocities. Therefore DWA is preferred for rescue missions but it needs an additional global path planner. During our research we found a strongly connected paper by Morales et al. [15-16], where they present a pure-pursuit technique that is based on a simple proportional controller using pose displacement as input error and the lookahead parameter for achieving the next given waypoint. Velocity commands are produced based on geometrical calculations of the needed radius for achieving the goal. Therefore, the presented algorithm does not consider a given waypoint and computes the best circular path towards the goal based on lookahead distance. The authors found the algorithm to behave extremely well and it was presented as the more stable and accurate for path following. Path following is a widely used and analysed technique. However, there is a lack of advancement made on these simple and robust algorithms, which is the main topic of this paper. The basic problem is following the given path from the start to the given goal based on an available mobile robot. Our goal is to create a highly adaptive path following algorithm that can be used with skid-steered mobile robot bases and is easy to re-use.

The presented paper is organized as follows. The second chapter describes the overall system in order for the readers to understand the autonomous system and the part we are focusing on. The next chapter describes in detail the proposed algorithm with its conditions and equations. The fourth chapter presents the simulation and the real experiment, with results. The conclusion and future work are provided in the last chapter.

2. Control System Overview

Our work is focused on USAR (Urban Search and Rescue) missions where the goal is to build mobile robots that will be able to assist rescuers in the event of disasters such as earthquakes, tsunamis and others. Rescue mobile robots

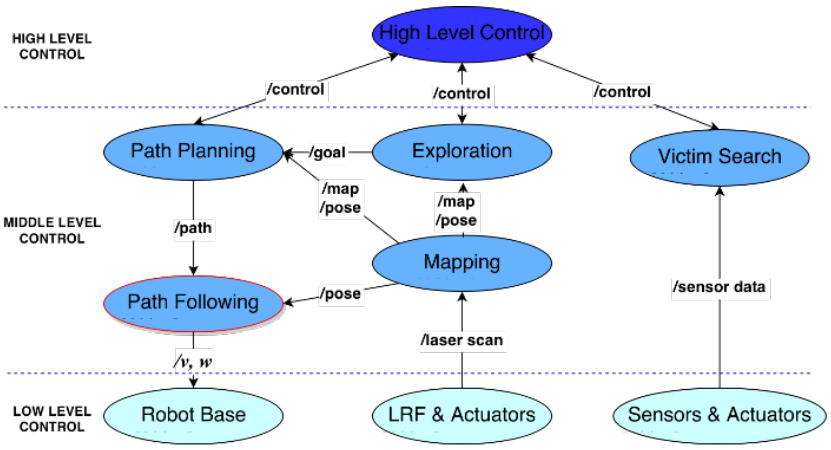

Figure 1. Autonomous rescue robot system overview

are expendable and they can reduce risks to rescuers. They provide a map and possible victims' locations and states. Rescue robots can be remotely operated but it is desired that robots be autonomous and, without any interaction, provide relevant information for rescuers. Based on the provided information the rescuers can plan safe rescuing actions and avoid unnecessary time spent during search actions. In the case of a disaster there is usually a lack of rescuers who can assist victims, therefore the rescue robot working autonomously can be of great use.

Various autonomous control systems for rescue mobile robots may have different system structures. A typical modular autonomous control system is described in this chapter. A modular system design is desirable because it can be easily adapted and provides a clear overview of the system's functionality. The general modular architecture of an autonomous control system, as shown in Figure 1, is taken from Siegwart and Nourbakhsh [17]. High-level control takes care of the system's behaviour where the middle level provides the system's functionality, which is controlled by high-level control. The low level is interfacing with actuators and sensors. Typically the low level communicates with the middle level, which provides preprocessing of sensor data and control commands of translational and angular velocities. The simplified graph in Figure 1 points out the path following functionality that is integrated just above the low-control level. In the case of a rescue, the mobile robot is placed within the unknown environment without a map. The rescue robot explores unknown space; therefore a map is provided by a real time mapping algorithm. A map world coordinate system (X, Y) is established at the robot's start. This map is provided by a mapping algorithm and path following is triggered when a path is provided from the system. The path is usually constructed over two stages; the first exploration provides a new global goal within a given space. In the second stage, the path planner constructs a path from the current robot position to a given goal within the environment. The constructed path is collision-free and is built based on the current map state. The mapping node provides updated information regarding the environment (mapping) and robot pose (position and orientation) within the environment (robot localization) on the basis of laser range finder scans and other sensors. The rescue robot position control loop is thereby closed multiple times within the autonomous rescue robot control system in order to handle perturbations. The Simultaneous Localization And Mapping (SLAM) algorithm [2] is the crucial component of the control system. SLAM also provides the position update (pose) for the path follower. As depicted in Figure 1, the prerequisites of the path following are a safe path in the environment from the path planner, updated robot pose from mapping and velocity control of a mobile robot base.

The mobile base is controlled by translational and rotational velocities that are calculated by the path following node. The mobile base (low-level robot base) control depends on the type of robot locomotion. The translational and rotational velocities are first transformed into track/wheel speeds and then used as reference values for PID-controlled robot driving motors. Robot velocities and robot position in space in the case of our tracked rescue robot are presented in Figure 2.

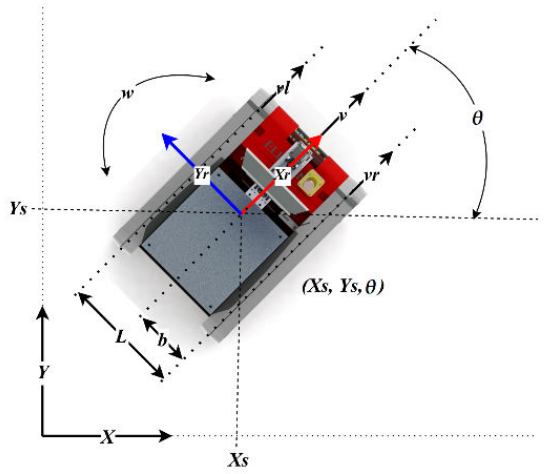

Figure 2. Mobile robot velocities and robot position presentation

An approximate kinematic model of a wheeled or tracked mobile robot was discussed in [18], where the tracked robot's kinematic, motion properties and problems are presented. The overall translational velocity v is defined in (1) and the rotational w in (2), where L is distance between wheels or tracks and is equal to $2b$. Velocity transformation for each track or wheel is calculated in (3) and (4).

$$v = \frac{vr + vl}{2} \tag{1}$$

$$w = \frac{vr - vl}{2b} \tag{2}$$

$$vl = v - \frac{w*2*b}{2} = v - w*b \tag{3}$$

$$vr = v + \frac{w*2*b}{2} = v + w*b \tag{4}$$

If translational velocity is zero and rotational greater than zero, we can extract from (3) and (4) that velocity of the right wheel is equal to negative velocity of the left wheel or vice versa. If rotational velocity is zero and translational velocity greater than zero, then left and right velocities are the same. The left and right robot tracks are controlled by a PID speed controller. Therefore we could consider the left and right tracks' actual speeds are also given by equations (3) and (4). The left and right tracks directly interfere with the environment (ground). Slipping occurs between the tracks and the ground. Slipping is very undesirable but it is necessary for the skid-steering of mobile robots. The complex dynamics of a track slippage due to the track-soil interactions of mobile robots is discussed in [19]. Grounds on disaster sites are very unpredictable, therefore we are not relying on the odometry provided by the wheels' encoders. The compact form of the kinematics of the differential drive mobile robot can be modelled by the equation (5) from [17], where $\dot{X}_r$ and $\dot{Y}_r$ are the velocities in the direction of X and Y in the robot base frame.

$$\begin{bmatrix} \dot{X}_r \\ \dot{Y}_r \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \tag{5}$$

Differential drive robots are actually describing motion along a circular path, as shown in Figure 3, when the velocity control of (v, w) is implemented.

The circular motion has its Centre of Rotation (CR). Straight line motion is considered when the rotational velocity is close to zero or zero. The relation between the circular

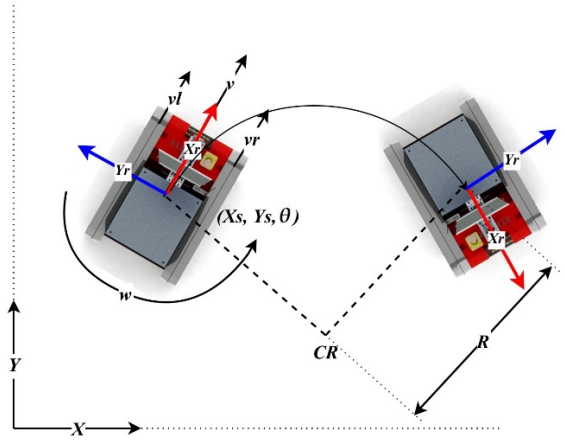

Figure 3. Circular movement formed, based on translational and rotational speed

movement radius R and the velocities on each track is defined in (6) from [7] and [19].

$$R = \frac{b}{2}\left(\frac{vr + vl}{vr - vl}\right) \tag{6}$$

If rotational velocity is zero ($vr = vl$) then in limit the circular path degenerates into a line; it is a circle with infinite radius. Therefore a line and a circle could be considered as mobile robot primitive motions. Common path follower algorithms include collision checking at low level. This ensures that the robot does not collide with dynamic obstacles or walls in the environment. The collision checker can be included in the system as well.

3. Path Following Control Method Using Arc Fitting

The DAF path follower focuses on finding correct velocities (v, w) for a mobile robot base based on the given path and robot pose update in order to achieve the best path following performance. An example of the given path is shown in Figure 4 above. It is desired that the robot drives along the given path within some reasonable tolerances. Experiments with our tracked rescue robot using the Hector path follower have shown that this is a real challenge. Figure 4 below shows a map with dark blue walls and a path produced by our tracked rescue robot using the whole Hector mapping and exploration stack as well as the whole Hector navigation stack (including Hector path follower). In Figure 4 below we can see lots of changes in robot orientation produced by the robot turning almost on the spot. Due to frequent turning, the robot translational motion is slow and, taking into account the narrow corridors of the arena, collisions with the arena walls also frequently occur. Therefore we found the performance of the Hector path follower unsatisfactory for our rescue robot and decided to develop our own DAF path follower, which is represented in the following.

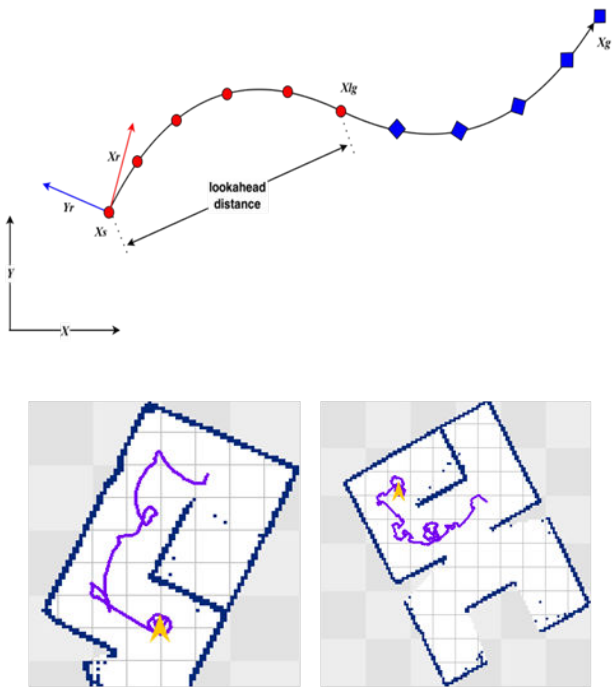

Figure 4. Given waypoints for mobile robot to follow (above) and below trajectories (violet) produced by the Hector_path_follower

In order to transform positions (waypoints) into command velocities, as shown in equation (7), we need a start and an end point or a global goal point. A start point is noted as $Xs(x_s, y_s)$ and a global goal as $Xg(x_g, y_g)$. Waypoints (x_n, y_n) are forming a given path and n is the number of provided waypoints.

$$\begin{bmatrix} v \\ w \end{bmatrix} = f(x_n, y_n) \tag{7}$$

In Figure 4 above the start point Xs, the global goal Xg and the local goal $Xlg(x_{lg}, y_{lg})$ are shown. The start point is updated by each robot pose update in order to compensate the track slipping on the ground and other perturbations. The local goal is defined by the distance lookahead parameter. The distance from the robot's position to the local goal corresponds to the lookahead distance parameter in [15-16]. The distance setting affects the trajectory of the robot; the shorter the distance, the more exact the following. The red circular waypoints present given points on the path which are considered as the local path on which the arc fitting is applied. The blue rhomboid waypoints are points on the given path, which lead to the global goal. In the next subsection we present the arc fitting algorithm, where our goal is to calculate the radius of a circle that approximates a given local path and control velocities (v, w) that would drive the robot on the circle approximated local path.

3.1 Arc Fitting

For our proposed algorithm of arc fitting we need points of the local path, as shown in Figure 5. The local path should include at least one waypoint. The lookahead distance cannot be larger than the distance to the global goal. If start point Xs is equal to local global point Xlg then the arc fitting algorithm is not performed. The radius R calculation is made based on the geometrical properties of circles and triangles, as can be seen in Figure 5. We need the circle height h, which is calculated in equation (8), to calculate R. The height is measured from the centre line that is formed by Xs and Xlg. The triangle area theorem is used to calculate the heights of the triangles for each given point along the local path. We use maximum height to get the best fitting of the circle. A_n is calculated from Heron's formula in (9). The semi perimeter or half of the triangle's perimeter ss is calculated (10).

$$h = \max_n \left(\frac{A_n * 2}{sideC} \right); h \geq 0 \tag{8}$$

$$A_n = \sqrt{ss(ss - sideA)(ss - sideB)(ss - sideC)} \tag{9}$$

$$ss = \frac{(sideA + sideB + sideC)}{2} \tag{10}$$

The variables $sideA$, $sideB$ and $sideC$ (11-13) are distances between the given start point, a point on a local path and a local goal point, which forms a triangle.

$$sideA = \sqrt{(x_s - x_n)^2 + (y_s - y_n)^2} \tag{11}$$

$$sideB = \sqrt{(x_n - x_{lg})^2 + (y_n - y_{lg})^2} \tag{12}$$

$$W = sideC = \sqrt{(x_s - x_{lg})^2 + (y_s - y_{lg})^2}; W \geq 0 \tag{13}$$

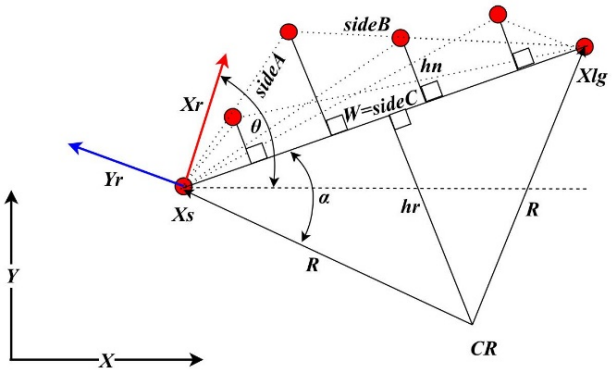

Figure 5. A local path example with waypoints and geometric model for calculating fitted circle

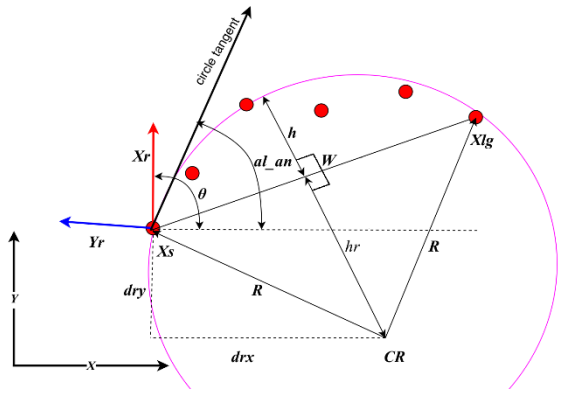

Figure 6. A local path example with fitted circle and robot alignment

The lookahead distance or W is equal to the distance between the start point and the local goal point (13), which is calculated as distance *sideC*. This distance is also used to calculate the height of an isosceles triangle, which is formed from the centre of the rotation point, the start point and the local goal point. We calculate the fitted circle radius R based on the geometrical properties. As shown in Figure 6, the sum of two triangles' heights given in (14) present the radius R of the fitted circle.

$$R = h + hr \tag{14}$$

We derive hr from triangle area base formula (15). For the calculation of an isosceles triangle A_r area we use (16).

$$A_r = \frac{1}{2}Whr \rightarrow hr = \frac{2A_r}{W};\ hr \geq 0 \tag{15}$$

$$A_r = \frac{W}{4}\sqrt{4R^2 - W^2} \tag{16}$$

Equations (15) and (16) are used in (14) to derive the circle radius R as shown in (17). Radius R is always positive.

$$R = \frac{h}{2} + \frac{W^2}{8h} \tag{17}$$

The given path can be very diverse and may have diverse heights hn in particular waypoints on the path (see Figure 5). Therefore the maximum height of h (8) is not necessary in the middle of the lookahead distance. The obtained fitted circle presents a smooth trajectory based on given local waypoints. In order to generate the correct sign of rotational velocity w the circle's centre is needed. The circle's centre is calculated from three given points $Xs\ (x_s,\ y_s)$, $Xlg(x_{lg},\ y_{lg})$ and maximum calculated height point $Xh\,(x_h,\ y_h)$. Equations (18-21) taken from [20] define the circle's centre $(Cx,\ Cy)$ or CR in the world coordinate system.

$$Cx = \frac{F1\left(y_{lg} - y_s\right) - F2(y_h - y_s)}{2\left(x_h - x_s\right)\left(y_{lg} - y_s\right) - 2(x_{lg} - x_s)(y_h - y_s)} \tag{18}$$

$$Cy = \frac{F2\left(x_h - x_s\right) - F1(x_{gl} - x_s)}{2\left(x_h - x_s\right)\left(y_{lg} - y_s\right) - 2(x_{lg} - x_s)(y_h - y_s)} \tag{19}$$

$$F1 = x_h^{\,2} - x_s^{\,2} + y_h^{\,2} - y_s^{\,2} \tag{20}$$

$$F2 = x_{lg}^{\,2} - x_s^{\,2} + y_{lg}^{\,2} - y_s^{\,2} \tag{21}$$

The sign of rotational velocity w is determined by a 2D analogue of cross product of vectors *XsXlg* and *XsCR*. *XsXlg* is a vector directed from point *Xs* to the point *Xlg* and *XsCR* is a vector directed from point *Xs* to the centre of circle *CR*. The result of the 2D cross product maintains geometrical meaning of cross product and indicates the sign of angle α between the vectors (Figure 5). If the cross product is positive then the sign of angle α is positive and vice versa. Figures 5 and 6 show a case of right turn where $w < 0$ because the angle α between *XsXlg* and *XsCR* is negative. In the case of a left turn the rotational velocity $w > 0$, because the angle α between *XsXlg* and *XsCR* is positive. Calculation of the velocity commands magnitude $|w|$ is done based on the user's given maximal translational velocity and radius calculation R. In our case we used half of the user-defined maximum speed v_{max} for translational velocity (22).

$$v = \frac{v_{max}}{2} \tag{22}$$

$$|w| = \begin{cases} w_{max}; if\ R < R_{min} \\ \dfrac{v}{R}; if\ R_{min} < R < \infty \\ 0\ ; if\ R = \infty\ and\ h = 0 \end{cases} \tag{23}$$

The rotational velocity magnitude |w| is defined by (23), where w_{max} presents a maximum allowable speed and R_{min} presents minimum radius set by the user. In practice the calculation of value R never exceeded 100 m. The velocity calculations are triggered every time a position update is available. This ensures that we have updated circular motion in real-time. A higher position update rate provides a more exact following. The path following is stopped when the robot reaches the Xg.

3.2 Orientation Adaptation

The tracked robots as our rescue robots are differential driven and non-holonomic; therefore the robot needs to be aligned with a tangent on a given circle before executing the circular motion. This step is necessary for the robot to perform the desired circular path. The alignment angle al_an (see Figure 6) is the angle between the circle tangent and x axis of the world coordinate system and is calculated using equation (24). The calculation of the alignment angle al_an is based on the line connecting the circle centre and the robot start point Xs, which is actually the circle normal at the start point Xs. In order to obtain the alignment angle al_an the angle between the circle normal at the start point Xs and x axis of the world coordinate system is increased by 90 degrees.

$$al\_an = atan2\left(\frac{dry}{drx}\right) + \frac{\pi}{2} \tag{24}$$

A new alignment angle is calculated at every new position update. Then the robot orientation angle and calculated alignment angle are wrapped to the interval [–PI to PI]. Figure 6 shows the robot's local path and the resulting circle, which fits the local path based on the calculated radius R. The desired robot orientation is the alignment angle al_an. The robot has its own orientation within the space, which is not necessarily the same as the alignment angle. The misalignment between the desired robot orientation and the actual robot orientation during the robot's motion appears to be due to the presence of ground irregularities or due to the robot slipping. In order to achieve good performance in the real world we defined three scenarios depending on the robot's actual orientation angle θ and the desired robot orientation angle al_an. Two threshold parameters an_th1 and an_th2 are needed to define three possible scenarios while the robot is in motion:

1. $\boldsymbol{abs(\theta - al\_an) < an\_th1}$:

The difference between the robot actual orientation and the alignment angle is so small that additional correction of the robot orientation is not necessary.

2. $\boldsymbol{(abs(\theta - al\_an) > an\_th1)}$ &
$\boldsymbol{(abs(\theta - al\_an) < an\_th2)}$:

The robot actual orientation angle is larger than the lower threshold and still smaller than the upper threshold. This is the case where we compensate the robot orientation during motion. The compensation is calculated using (25) and the result is added to the rotational velocity. This affects the performance of the resulting trajectory. Parameter t_l presents the time period on which the local path is calculated and is defined by the user. To reduce the number of parameters we use only t_l parameter, from which we also calculate the lookahead distance as multiplication of t_l and the maximum allowable speed also given by the user.

$$w = w + \frac{\theta - al\_an}{t_l} \tag{25}$$

3. $\boldsymbol{abs(\theta - al\_an) > an\_th2}$:

In this case the deviation of the robot actual orientation from the desired orientation is too big to be compensated for during the robot motion, therefore our rescue robot is stopped and since it is a differential drive robot it turns on the spot until the alignment orientation is reached.

In order to achieve the best possible performance in the real world we also implemented a compensation for uneven ground, as described in the next section.

3.3 Uneven Ground Compensation

Mobile rescue robots are often placed within a complex environment, where the robot needs to overcome ramps, stairs or other obstacles. The robot behaviour changes on uneven ground. Here we propose a ground compensation algorithm that modifies the current local circular path of the robot based on ground inclination. The dynamically fitted circular path is calculated in subsections 3.1 and 3.2. This path is calculated in 2D space, where no inclinations are considered. The projection of the circle on inclined ground gives an ellipse that is different from the calculated circle. A prerequisite for this improvement is information on the robot orientations over axes: roll defines robot rotation in X axes, pitch rotation in Y axes and yaw is the robot's rotation in Z axes. Roll, pitch and yaw are usually provided by the Inertial Measurement Unit (IMU). In our case we use IMU that has integrated signal processing. The sampling is done at high frequency, a low pass filtering and Strap Down Integration (SDI) at 2 kHz provide Altitude and Heading Reference Systems (AHRS) information. In order to correct local trajectory we considered the robot rotations and modified triangle parameters W and h.

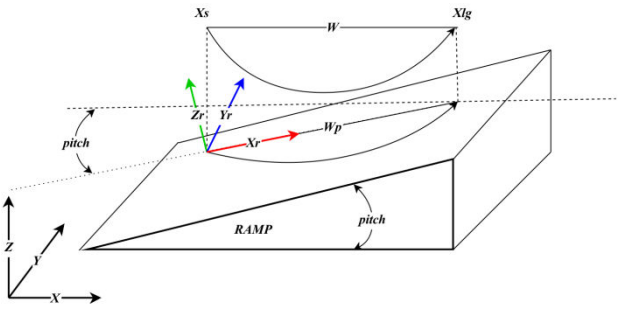

Figure 7. Mobile robot path following on a ramp

$$Wp = \frac{W}{\cos(pitch)} \tag{26}$$

$$hr = \frac{h}{\cos(roll)} \tag{27}$$

Adapted parameters influence the local circular path calculation and improve the performance of the robot on uneven ground. An example of the robot on a ramp with pitch inclination is shown in Figure 7. A projection of an arc on inclined ground causes deformation of the circular path, which we corrected according to equation (26). The same calculation is done if the roll inclination is present as given by (27).

4. Simulation and Experimental Results

Our path following algorithm was implemented in ROS, which is popular throughout the robotics community. To verify the performance of the DAF path following algorithm we conducted simulation and experimental tests in the real world. For simulation we used a ROS Stage [21] simulator and for real experiments we used our rescue robots. We placed the robot within different environment scenarios and test algorithm performances. Our rescue robot hardware setup is described in this section.

4.1 Simulation Setup

The experiments conducted as simulations were carried out based on the virtual models of our rescue robots and the Stage simulator. A model of the robot can be seen in Figure 8 (left). The simulation was performed on flat ground and it did not include ramps. For the simulation we used our control system, as described in the second section.

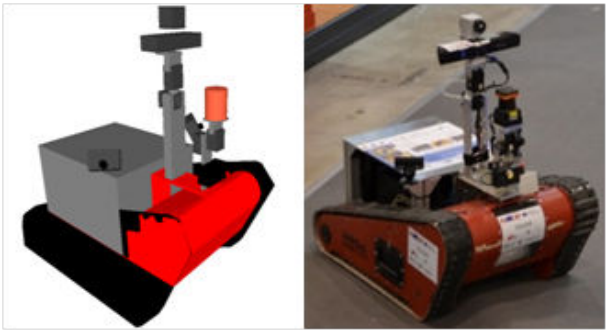

Figure 8. Model of rescue mobile robot Mesa Element in simulation (left) and real robot setup (right)

Comparisons between the DAF path follower, the move_base, and the Hector path follower were done during the simulation using the same environmental setup. The simulated environment was constructed in such a way that the robot can only progress along one designated path. Our first choice was the move_base DWA implementation of the path follower because it is available in ROS. Simulation results in the form of a map and robot trajectories are shown in Figures 9, 10 and 11. The light grey squares present 0.5 × 0.5 m area and the yellow arrow presents the global coordinate system, which is initialized when mapping is started.

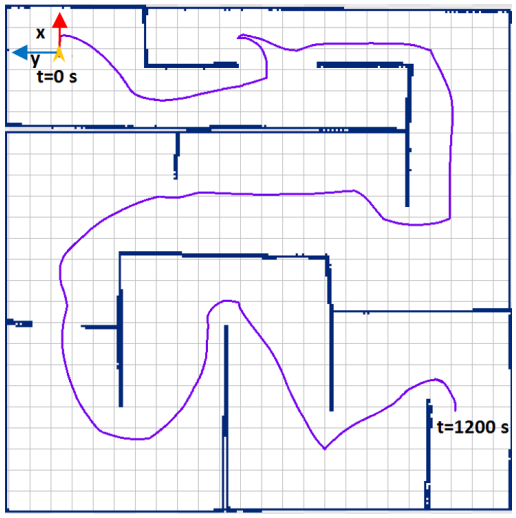

Figure 9. Map and trajectory produced by move_base (DWA) in simulation

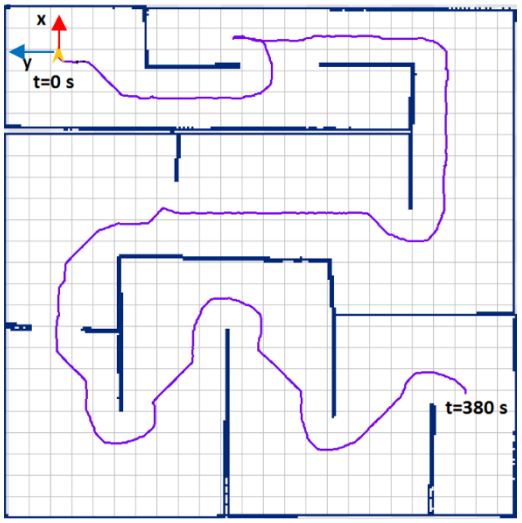

Figure 10. Map and trajectory produced by Hector software in simulation

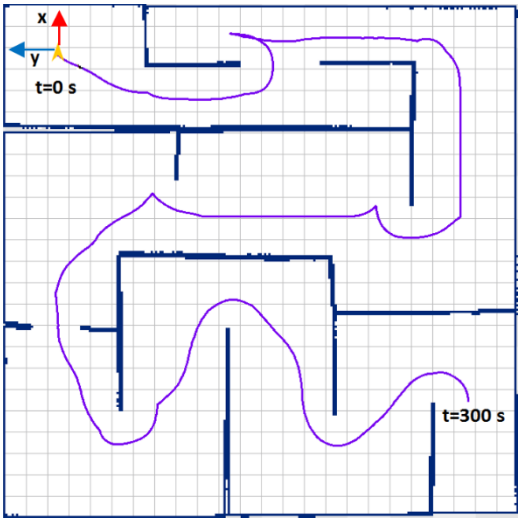

Figure 11. Map and trajectory produced by DAF path follower in simulation

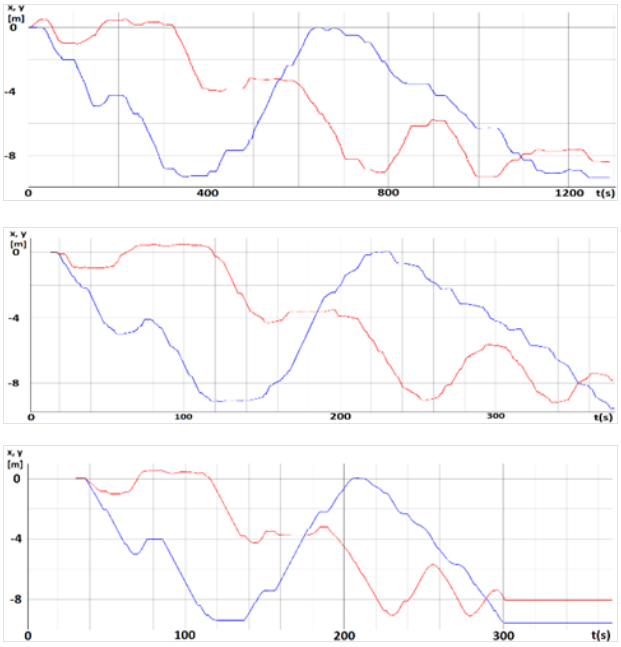

Figure 12. Position change over time for move_base (DWA) (above), Hector (middle) and DAF path follower (below) where the red graph is robot X position and the blue graph is robot Y position in metres

The resulting map from the move_base path follower is shown in Figure 9. The resulting paths from the Hector and DAF path followers are shown in Figures 10 and 11. The resulting robot trajectory obtained using the Hector algorithm is similar to the DAF path follower's but is more rough and serrated. All the path followers are performing really well in simulation. We would like to stress that use of the DAF path follower has resulted in a continuous path with the skid-steered mobile robot. Differential or skid-steered robots can turn in place and they are able to drive on almost any circular trajectory, therefore there is no need for driving backwards. In the case of Ackerman steering, a negative velocity is necessary to correct robot orientation. A better overview of the differences between the described systems can be seen in Figure 12, which shows time dependence of the robot's position (x, y coordinates). Comparison of coordinate graphs (Figure 12) shows that the DAF path follower exhibits the least ripple in position when compared with the move_base and Hector path follower. The ripple in position also affects the time performance of the path follower. The path following with the move_base has taken 1250 seconds in the simulated environment. The Hector path follower, compared with the move_base, completes the same task in 380 seconds and the DAF path follower needed 300 seconds. We should mention that the move_base incorporates collision avoidance. Therefore long calculation times are reasonable.

We wanted to check how the velocities were behaving over time, which is presented on the graphs in Figures 13-17. The translational and rotational velocity of the move_base over time is shown in Figure 13. We can notice that the speed limits are set quite high compared with the Hector and DAF path followers. Despite high velocities the move_base needed lots of time to complete its task. Figure 14 shows the translational velocities produced by the Hector path follower; we can see that the velocities were stirred up and had multiple negative translational velocity values, which means that the robot was travelling in reverse. There was also lots of stopping on the way. Figure 15 shows velocities produced by the DAF path follower. We can see that the translational velocity is constant during motion and changes to zero during stop. The changes in translational velocities were immediate, causing the robot to achieve the given speed in the minimum possible time. The same is valid for stopping. We can afford to do so because soft stop and go is usually implemented within the low-level motor controller. Noticeable are the two thresholds implemented in the Orientation Adaptation section, where a small deviation is present while the robot is using rotational velocity. The second threshold can be seen with the inclination of rotational velocity just before it goes into maximum.

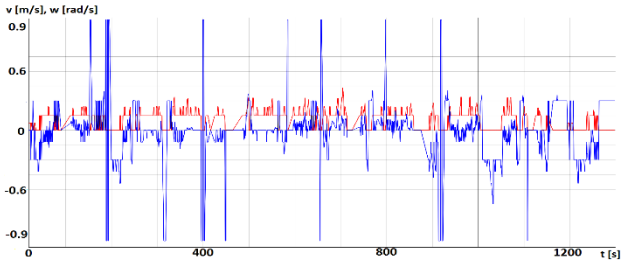

Figure 13. Move_base translational in (red (m/s)) and rotational (blue (rad/s)) velocity over time

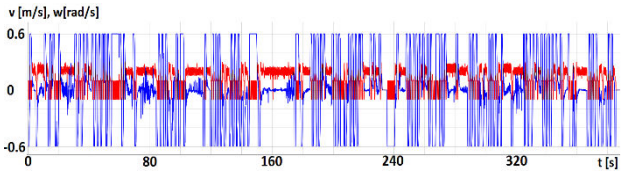

Figure 14. Hector translational (red (m/s)) and rotational (blue (rad/s)) velocity over time

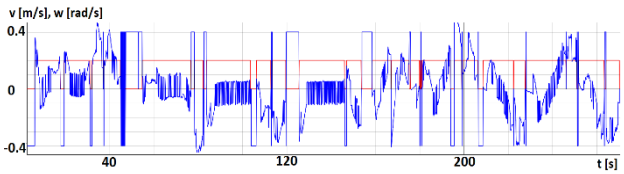

Figure 15. DAF path follower translational (red (m/s)) and rotational (blue (rad/s)) velocity over time

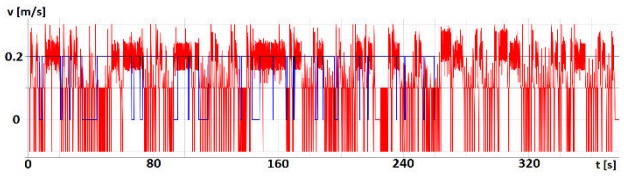

Figure 16. Translational velocity of Hector (red (m/s)) and DAF path follower (blue (m/s))

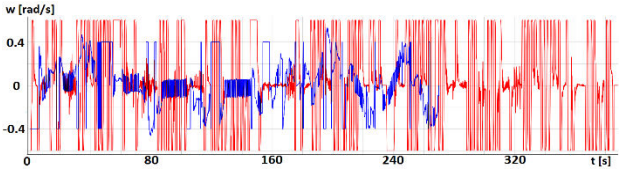

Figure 17. Rotational velocity of Hector (red (rad/s)) and DAF path follower (blue (rad/s))

The best performance regarding time and the fitting of the produced trajectory was achieved with the Hector_path_follower and the DAF path follower. Figures 16 and 17 show the comparison between the implemented translational and rotational velocities. In Figure 16 we can see that the translational robot velocity obtained by using the DAF path follower is predominantly constant, as expected due to equation (22). In contrast, the translational velocity obtained by using the Hector path follower changes velocity direction frequently. Because of frequent velocity changes the robot proceeds more slowly. The comparison of rotational velocities is made in Figure 17, where we can notice bigger changes in velocities on the red graph compared with the blue. Smaller perturbations in rotational velocity produce smoother turns and the robot does not oscillate during motion.

The DAF path follower has parameters (described in 3.1 and 3.2) that can be defined by the user. The parameter setting affects the robot's following behaviour and fitting of produced trajectory to the given path. Figure 18 shows three different trajectories produced by the DAF path follower where we had adjusted the lookahead parameter.

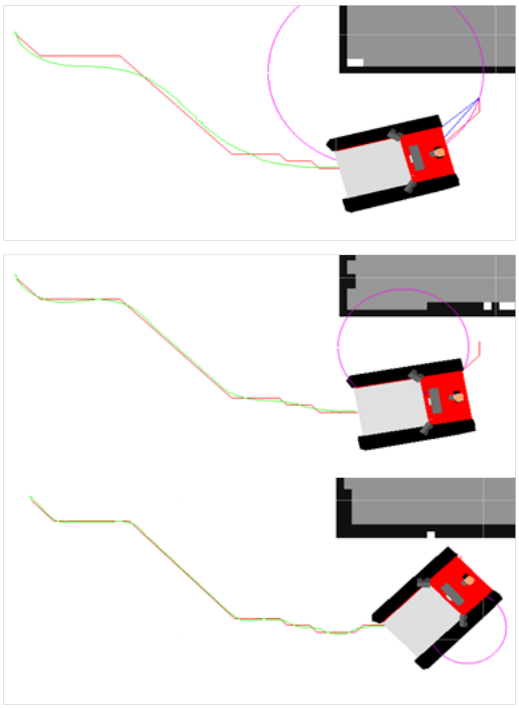

Figure 18. DAF path follower performance: (above) lookahead 2.0 m, (middle) lookahead 1.5 m, (below) lookahead 0.5 m

We can notice that the smaller lookahead parameter produced a more exact following, but the small curves are still noticeable at the sharp turns. The red line presents the path that was to be followed and the green line is the robot's trajectory. Figure 18 also shows auxiliary violet circles and in the blue colour a local triangle formation. We compared our DAF path follower to the Hector path follower shown in Figure 19. We can notice that the Hector path follower was executing a very exact following. In our case we can adapt the fitting of the produced trajectory to the produced trajectory and within exactness of the following.

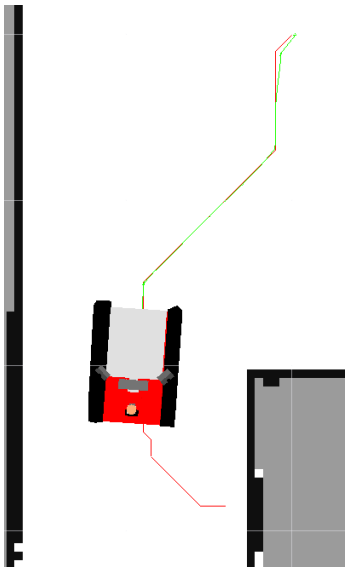

Figure 19. Hector path follower performance with default parameter settings

4.2 Experimental Setup

Our experimental setup included a standard NIST [22] rescue arena, which is used in RoboCup Rescue [23] competitions. Our rescue mobile robot, shown in Figure 8 (right), was built on a Mesa Element robot base with a tank-shaped track system, which is able to overcome stairs and carry loads of up to 55 kg. The robot was built using two mechanical and sensory systems of the laser alignment (LAS) and the sensor head system (SHS). The laser alignment system provides alignment of the Laser Range Finder (LRF) in two degrees of freedom. It is based on two smart servomotors from Robotis (Dynamixel RX-24F), which are integrated in mechanical holders with a LRF mounted on the top. The system can be seen in Figure 20 (left). The system's functionality is to provide stable horizontal alignment of LRF measurements and details of the implementations are described in [24]. That study also includes performance tests and research on response time of the system. This system ensures that we have relevant map information even if the robot traverses dynamic grounds. The second system of the sensor head is a combination of mechanical holders and smart servomotors (Dynamixel RX-64), which can be seen in Figure 20 (right). A collection

of sensors can be mounted on the sensor mounting plate, which is mounted on top of two servomotors. This adds additional coverage space to the sensors in two dimensions. The sensory system allows larger ground coverage while the robot is moving and increases the possibility of life signs detection. Our sensory system for detecting people consists of the thermal camera, CO2 gas detector, microphone and the Microsoft Xbox Kinect sensor, which provides depth image and colour image. For this experiment the LAS and the robot mobile base were used.

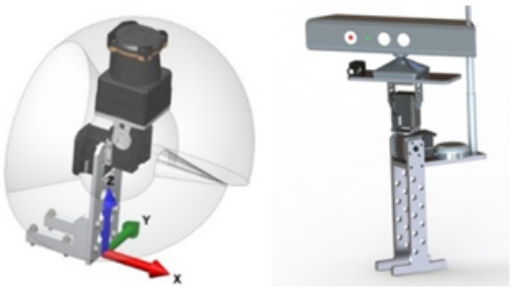

Figure 20. Rescue robot on-board systems, (left) laser alignment system and (right) sensor head system

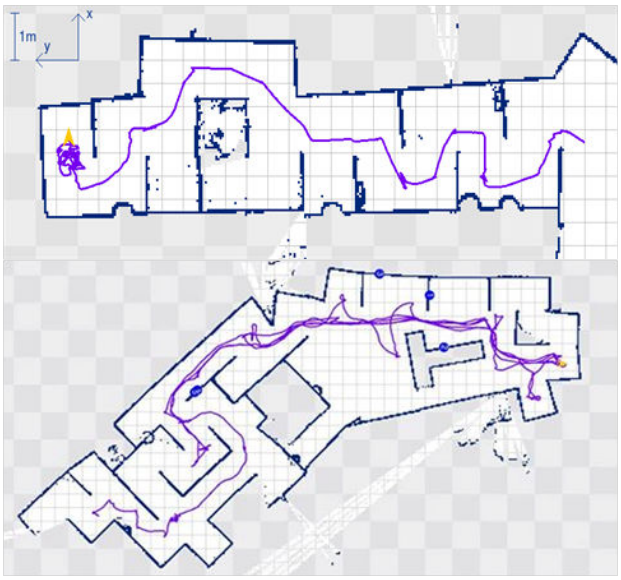

Figure 21. Above) part of maps and robot trajectory built at the RoboCup German Open 2014 competition, (below) map and trajectory built at RoboCup 2014 in Brazil

The DAF path following algorithm was tested at the RoboCup German Open Rescue robot competition where robots were competing in a standard NIST arena setup. Our first competitions were conducted with a move_base path follower where the results were not satisfactory. The robot was unable to successfully follow the given path in the structured NIST arena (see Figure 24 below). The path following with the Hector path follower setup also failed in real environments. We were unable to adapt given parameters to achieve a collision-free path following. The robot was hitting the walls of the arena. In our latest RoboCup competition we used a DAF path follower where the results can be seen in Figure 21. The dark blue lines on the map represent the walls in the arena, the yellow arrow is the start position, and the purple line is the produced robot trajectory. We can see that at the start, the robot performed some turns in a place that caused lots of slipping because of its complex ground structure. The mobile robot on the given irregular ground at the RoboCup German Open 2014 competition can be seen in Figure 22. While the robot is rotating, it drifts away unpredictably because of the ground structure. Our algorithm corrected the robot's position while executing the translational movements. The DAF path following algorithm was able to follow the given path and we were able to achieve an autonomous run using a tracked skid-steering mobile robot, which fulfilled our expectations.

Figure 22. NIST arena setup at RoboCup German Open 2014

The next large test of our algorithm was performed at the world RoboCup 2014 competition in Brazil. Figure 21 below shows the results from the RoboCup 2014 competition where our robot competed in the autonomous challenge and it explored the largest area autonomously. During the run three soft restarts were performed; that is why there are multiple paths shown on the map.

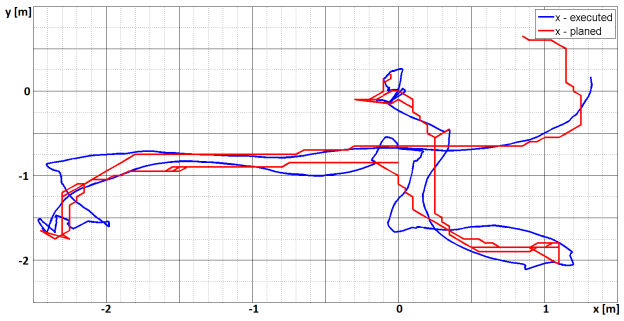

Figure 23. A full area of robot run in NIST arena (red – planned path, blue – actual robot path) in real experiment

In order to evaluate the performance of the DAF follower in greater detail, we recorded the data during a real experiment in a real NIST arena. Our robot control system is designed to calculate a new section of robot path every 15 seconds because the goal is to explore as much space as possible. The planned path recalculation is performed by the exploration controller and is based

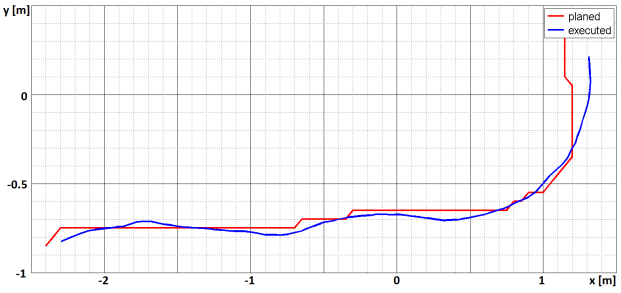

Figure 24. A section of robot run in NIST arena (red – planned path, blue – actual robot path) in real experiment

on time or proximity of the global goal. Figure 23 shows the planned path in red and the actual robot path in blue. In Figure 23 we can notice that the planned path has multiple routes due to recalculation. To display the results of the experiment more clearly, we took a section of the path as shown in Figure 24. Figure 24 shows an approximately 3.5 m long section of robot run performed with maximum translation velocity 0.4 m/s, maximum rotational velocity 0.2 rad/s, an_th1 0.15 rad, al_an2 0.9 rad and lookahead parameter 0.8 m. The parameter setup was the same as in the RoboCup competition in Brazil (see Figure 21 below). An actual path (blue) deviation from the planned path (red) is shown in Figure 24.

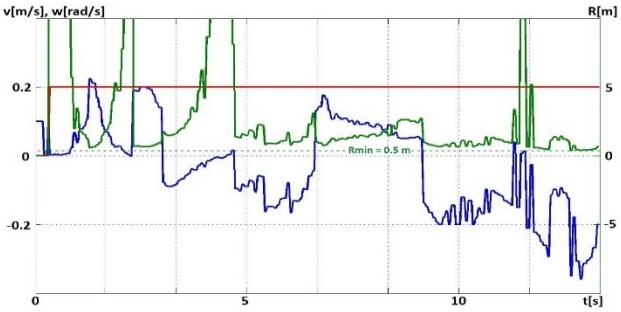

Figure 25. DAF path follower translational (red), rotational (blue) velocity and computed R (green, where values are shown on right Y axis) over time in real experiment

The blue line in Figure 24 smoothly follows the red line. The ground surface where we performed the experiment was flat and slippery. The real experiment shows how the robot drifts away while turning. The deviations are the result of velocity error, execution errors, unpredictable ground and loose parameters settings. The maximum deviation error produced on X axes is 0.28 m and the maximum error on Y axes is 0.12 m. Figure 25 shows a time evolution of calculated translational and rotational velocity, where we can see constant translation velocity and active area of rotational velocity. The green line shows evolution of the calculated radius R, where minimum $Rmin$ is shown. The upper value of R is limited only by its own numerical precision. Despite all perturbations the DAF path follower has followed the given path.

5. Conclusion

This research work presents a path following algorithm that is very adaptable to the different skid-steering mobile platforms. The algorithm has an easy setup and it considers ground inclination while following a path. The fitting of a given path is performed with circular objects. The proposed algorithm can perform continuous path execution based on the waypoints that are given by the provided path. The algorithm was tested within a simulation environment and at the RoboCup German Open competition. With this proposed algorithm we tried to fill the gap between simulation and the real robot application. This algorithm will be used on our rescue robot at the RoboCup Rescue robot competition in Brazil 2014. Our DAF path follower is also available in ROS repositories under the name tedusar_daf_path_follower [25] as open source software. In future work we will merge our current work in path following with a 3D planning algorithm, which is currently in development. Our goal is to provide a complete navigation system for complex indoor scenarios such as disaster sites.

6. Acknowledgements

The work has been partly funded by the European Fund for Regional Development (EFRE), the Land Steiermark and the Republic of Slovenia under the TEDUAR grant.

7. References

[1] Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, Wheeler R and Andrew N. ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software, Willow Garage, Menlo Park, California. 2009.

[2] Kohlbrecher S, Meyer J, Klingauf U, Stryk O. A Flexible and Scalable SLAM System with Full 3D Motion Estimation. In: IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR); 1-5 November 2011. p. 155-160.

[3] ROS hector_navigation stacks [Internet]. Updated on 16 May 2014. Available from: wiki.ros.org/hector_navigation. Accessed on 18 Apr 2014.

[4] Choset H, M. Lynch K, Hutchinson S, Kantor G, Burgard W, E. Kavraki L and Thrun S. Principles of Robot Motion. Cambridge, Massachusetts. A Bradford Book, The MIT Press. 2005. p 440-472.

[5] Lamiraux F and Laumond J. P. Smooth Motion Planning for Car-Like Vehicles. In: IEEE Transaction on Robotics and Automation, Vol. 17, no. 4. 2001.

[6] Laumond J. P, Jacobs P. E, Taix M and Murray R. M. A Motion Planner for Nonholonomic Mobile Robots. In: IEEE Transaction on Robotics and Automation, Vol. 10, no. 5. 1994.

[7] Yazici A. A Smooth Tour Construction Approach for a Mobile Robot with Kinematic Constrains. In:

International Journal of Advanced Robotic Systems, Vol. 10. 2013.

[8] Lee S and Hyeon J Park. Dynamic Path-Following Using Temporary Path Generator for Mobile Robots with Nonholonomic Constraints, Mechatronics Lab., Precision Mechanics Eng. Hanyang University, Seoul, South Korea. 1999. p. 631-636.

[9] F. Diaz del Rio, G. Jimenez, J. L. Sevillano, S Vicente, A. CivitBalcells. A Generalization of Path Following for Mobile Robots, Facultad de Informatica, Universidad de Sevilla, Spain. 1999. p. 7-12.

[10] Indiveri G, Nuchter A and Lingermann K. High Speed Differential Drive Mobile Robot Path Following Control With Bounded Wheel Speeds Commands,. In: IEEE International Conference on Robotics and Automation, Roma, Italy. 10-14 April 2007.

[11] Komoriya K and Tanie K. Trajectory Design and Control of a Wheel-type Mobile Robot Using B-spline Curve. In: IEEE International Workshop on Intelligent Robots and Systems. Tsukiba, Japan, 4-6 September 1989.

[12] ROS pose_follower software package [Internet]. Updated on 6 May 2011. Available from: wiki.ros.org/pose_follower. Accessed on 18 Apr 2014.

[13] Gerkey P. B, Konolige K. Planning and Control in Unstructured Terrain, Willow Garage, SRI Artificial Intelligence Centre. 2008.

[14] Fox D, Burgard W and Thrun S. The Dynamic Window Approach to Collision Avoidance. In: Robotics & Automation Magazine, IEEE, Vol.4, no. 1. 1997. p. 23-33.

[15] Morales J, Martínez L. J, Martínez A. M and Mandow A. Pure-Pursuit Reactive Path Tracking for Nonholonomic Mobile Robots with a 2D Laser Scanner, Hindawi Publishing Corporation EURASIP Journal on Advances in Signal Processing, Volume 2009.

[16] Amidi O. Integrated Mobile Robot Control, The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania. 1990.

[17] Siegwart R, Nourbakhsh I. R. Introduction to Autonomous Mobile Robots, London, England, A Bradford Book, The MIT Press. 2004.

[18] Kelly A. An Intelligent Predictive Controller for Autonomous Vehicles, The Robotics Institute Carnegie Mellon University, 2 May 1994.

[19] Martínez J. L, Mandow A, Morales J, Pedraza S, García-Cerezo A. Approximating Kinematics for Tracked Mobile Robots,. In: The International Journal of Robotics Research, Vol. 24, no. 10, October 2005, p. 867-878.

[20] Technical Report for the year 1986 (in Slovene language), Poročilo o Delu za Leto 1986, Robotizacija, 03-2671/796-86, University of Maribor, Maribor 1986.

[21] Owen J, Nickels K. How to Use Player/Stage, 3rd Edition, Computer Science Department, Trinity University, San Antonio, Texas. 15 July 2013.

[22] Jacoff A, Messina E, and Evans J. Experiences in Deploying Test Arenas for Autonomous Mobile Robots. In: Proceedings of the 2001 Performance Metrics for Intelligent Systems (PerMIS) Workshop. 2001.

[23] RoboCup Wiki [Internet]. Updated on 28 March 2014. Available from: en.wikipedia.org/wiki/RoboCup. Accessed on 07 Apr 2014.

[24] Lepej P, Maurer J, Steinbauer G, Uran S. Analysis of laser sensor system for a rescue robot. In: Zajc B, Trost A, 22nd International Electrotechnical and Computer Science Conference ERK 2012. Portorož, Slovenia, IEEE Region 8. 17-19 September 2012. p. 127-130.

[25] Dynamic Arc Fitting path following algorithm. Available from: wiki.ros.org/tedusar_daf_path_follower. Accessed on 18 May 2014.