Improvement of an Adaptive Fuzzy-Based Obstacle Avoidance algorithm using Virtual and Real Kinect Sensors

György Csaba*
Óbuda University, John von Neumann Faculty of Informatics, Hungary
*csaba.gyorgy@nik.uni-obuda.hu

Abstract—The article presents improvements to an already existing fuzzy-based navigational system that was already described in a previous paper. To determine the movement direction of the robot, that system used a rule set that contained 16 rules based on 3 distance measurements [1], but this system was not driving the robot in a straight line when navigating in narrow corridors. To get rid of this error, the fuzzy deduction systems were redesigned and new adaptive field of view and new throttle control were introduced; the results of these improvements are that the robot now moves faster in obstacle-free environments and follows a straighter path in narrow corridors. The developed system uses 3 distance measurements as its input to generate the field of view and determine the movement direction for the robot and generate the reference signal for the expected speed of the robot. To calculate these values, three Mamdani-based fuzzy controllers are used that contain 7, 18 and 3 rules, respectively. Thanks to the new system design, the cumulative error of the sensors and the actuators is decreased, because the vehicle makes less turns when it is between straight walls; and since the curve of these turns are smaller as well, these errors will be smaller than with the previous system. For this reason, the estimation of the robot’s location and the creation of the environmental map (SLAM¹) are more accurate as well. To compare the different navigational methods in an impartial and unbiased way, there was a need to develop a robot and a Kinect sensor simulator, the main development steps of those are described as well. The final system was tested on the simulator and in a real-life environment as well, and the tests showed that the robot can successfully perform the obstacle avoidance and other navigational tasks as well. At the end of the article, the results of the various implemented sub-tasks (e.g. the adaptive field of view and the throttle control) are compared and this shows that the final algorithm works ideally in real-world circumstances; this is mainly noticeable when driving through narrow corridors.

I. INTRODUCTION

There are several possible solutions in the field of obstacle avoidance and real-time navigation, but usually these approaches can only be used if we accept some compromises ([2], [3]). Most of these problems can be avoided if we use techniques of soft computing ([4], [5]), but then new questions can arise (e.g. the number of layers if we use machine learning algorithms [6]). Most of the resources that describe fuzzy-based robot controlling systems deal with rule sets for two-wheeled vehicles ([7], [8]); controls like this are already being used in traditional systems since quite a while now ([9]). A different widespread method deals with robots with multiple wheels where the axles cannot turn ([10]), this type of control is currently used mostly with vehicles that move on caterpillar tracks. There are also two-legged humanoid robots ([11]); and four-legged and six-legged robots as well ([12], [13]). An example for a fuzzy-based robot control with front-axle steering is shown in article [14], but there is no available generally accepted solution for a fuzzy-based robot control using two steered axles. The approach described in [14] uses a route planning method that is similar to the one implemented by me (which is a Potential Field-based route planning (EPF) combined with a feedbacked fuzzy rule set. The outputs of the controller are two correctional coefficients, one for the steering and one for the acceleration); but there (contrary to my solution) only the first wheel is steered.

The fuzzy-based obstacle avoidance algorithm that was described in one of my previous articles ([1]) implemented an efficient real-time method for avoiding obstacles and navigation in large open areas, but in narrow corridors it drove unnecessarily too close to the wall on the left hand side, and then it avoided the collision with a sharp right turn (when the frontal distance changed to “near”). For this reason, it was necessary to upgrade this algorithm with the navigation in narrow corridors in mind. This development can be split up into three main parts: firstly the rule set was re-written and extended; then adaptive throttle control was introduced with the distances of the obstacles taken into consideration. Thirdly, a new adaptive rule set to determine the field of view of the obstacle detection was developed that is based on the speed of the robot and the distances in the front and on the sides. The newly developed adaptive system was created using three components: one Mamdani-based obstacle avoidance fuzzy controller that is made of 18 rules and uses three input and one output variables; one deduction subsystem that is made of 7 rules and uses three input variables to detect the obstacles and determine the angles; and finally a controller to determine the speed of the robot based on three rules. The resulting navigational system produces a better route-planning and obstacle avoidance algorithm, because depending on the distances of the obstacles the robot uses different movement speeds. During the navigation, it is possible to use the sensor data to build up two-dimensional ([15]) and also three-dimensional ([16]) environmental maps. The results of

¹Simultaneous Localization and Mapping
the fuzzy navigational system also affect the processes used for localization and map building (SLAM) as well, because (due to the less number of turns) the cumulative errors grow slower than they did before. In addition to this, thanks to the adaptive control of the field of view that is now used, the robot navigates itself through an unknown area faster than before (when it used only simple throttle control mechanisms) (see table IV.).

The navigation was tested using a mobile robot developed by us that was already described in a previous article of ours ([17]). The vehicle is a four-wheel drive RC car that is 40 x 30 cm², big and that has separate steering for the front and the rear axles as well. The communication between the software and the actuators is performed by an on-board electric circuit that was also developed by us (its block diagram can be seen on figure [17]). To detect the obstacles, we used Kinect and optical distance measurement sensors. To impartially compare the various approaches, it became crucial to develop a program that is capable of emulating the robot and the detector systems as well.

At the beginning of the article I describe the main policies and processes of the planning and the development phases of the emulator software; after this I explain the improvements of the fuzzy-based navigational system and I compare the new version to the old one. The developed systems were tested using both the MATLAB fuzzy simulator and a target-specific simulator program; and then using a robot car capable of driving autonomously as well. To detect the surrounding environment, real and virtual Kinect and infrared distance measurement sensors were used.

II. SIMULATOR

To impartially compare the newly developed and the already prepared systems firstly a sensor and robot emulator software had to be developed. The created software is also capable of a combined usage of real and emulated devices, the system design of the software is shown on figure 1. The main task of the navigation algorithm is to perform the obstacle avoidance using the image provided by the real or the simulated sensors, and also the communication with the robot or the robot simulator (the extensive description of this process can be found in the chapter titled Obstacle Avoidance and Navigation). The Kinect simulator uses a pre-defined binary global map to generate a sensor-detected image based on the location and the orientation of the virtual robot. The robot simulator (according to the received messages and considering the dynamic parameters of a robot with two steered axles (figure 3)) answers the controlling software with the definition of the relative movement of the simulated robot.

A. Kinect simulator

During the initialization of the Kinect simulator the program must receive a global top-side viewed map along with the location and the orientation of the robot (see figure 2(a)). According to the orientation of the robot, the simulator uses this global map to generate a surface map that would be detected by a Kinect sensor located on the simulated robot (on figure 2(b)) the green arrow shows the location and the orientation of the robot). During the design of the system, we gave special attention so that the new functions comply with the methods that can be found in the Kinect SDK. As a result of this, the previously developed programs that use the real Kinect devices can easily be rewritten to use the virtual Kinect sensor instead.

B. Robot simulator

The input parameters of the robot simulator are the speed and the steering angle (the location of the robot is not required, because only the motion differences that occurred between the function calls are transmitted from the robot to the controlling software, the analysis of this data is done by the controlling program). When querying the position of the robot, the motion values are reset and as a return parameter these motion values (the new direction and the distance of the robot) are returned to the caller function. To calculate these values, we have to know some of the mechanical parameters of the robot, such as the wheelbase and the orientation of the wheels. In this case, the new position of the robot can be calculated according to the followings (at low-speed turning):

- With front-wheel steered vehicles, the steering angle of the wheels can be defined using equations 1 and 2 (see figure 3).

\[
\varphi_{fl} = \tan^{-1} \frac{L}{R + t/2} \approx \frac{L}{R + t/2} \quad (1)
\]

\[
\varphi_{fr} = \tan^{-1} \frac{L}{R + t/2} \approx \frac{L}{R + t/2} \quad (2)
\]

Where \( \varphi_{fr} \) is the front right steering angle (in degrees), \( \varphi_{fl} \) is the front left steer angle (in degrees), \( R \) is the radius of turn and \( t \) is the wheelbase.

- The average angle of the front wheels is defined as the Ackerman Angle [18]:

\[
\varphi_f = \frac{L}{R} \quad (3)
\]
(a) The map that was given to the simulator ("R" signs the location of the robot, the red box shows the area detected by the sensor) and the area as it is detected by the virtual sensor.

Fig. 2. Demonstration of the operation of the Kinect simulator. (Green marks the location and the orientation of the robot.)

\[
\varphi_r = \xi \varphi_f \tag{4}
\]

\[
\varphi_r + \varphi_f = \varphi_f + \xi \varphi_f = \varphi_f (1 + \xi) = \frac{L}{R} \tag{5}
\]

Where \( \varphi_r \) is the rear right steering angle (in degrees) and \( \varphi_l \) is the rear left steer angle (in degrees).

\[
R = \frac{L}{\varphi_f (1 + \xi)} \tag{6}
\]

III. CREATING THE ENVIRONMENTAL MAP

To speed up the calculations performed on the map provided by the virtual or the real sensors (figure [1.10(a)], we perform a lossless compression and as a consequence the detected obstacles will be still kept intact [19] (figure [1.10(b)]).

To be able to compare the methods that we tried out, the path finding algorithm is still executed using the reduced map, but thanks to the Fuzzy systems, not even a bigger map would mean a significant loss in performance. Furthermore, when using this compression method, our approach remains usable [1] even despite the errors that occur in the detections of the sensors [20].

IV. OBSTACLE AVOIDANCE AND NAVIGATION

Our previous system ([1]) followed a straighter path than the algorithm using wave propagation algorithm ([17], [19]), but it ran a far from ideal course when moving between narrow corridors (it often oscillated when approaching the walls on the sides (figure 6(a))). To eliminate these errors, I applied three main methods:

- Definition of the speed depending on the distances from the obstacles.
- Redesign and augmentation the old rule set that contained only 16 rules.
- Adaptive definition of the obstacle detection’s field of view (viewing angle) depending on the distances from the obstacles.

Based on the principles specified in my previous article [1], I scaled the values of the input and output membership functions according to the dimensions of the reduced environmental map built by the sensors and according to the physical parameters of the robot. The fuzzification of the input distances is done using the same trapezoid membership functions for all three rule sets. According to this, all three inputs contain three trapezoid membership functions according to the different Fuzzy representations of the measured distances (figures [1.4(a) and [1.4(b)].

A. Speed Control

When making a turn before an obstacle, sometimes the refresh rate of the top-side viewed images meant a problem. If the robot was moving at high speeds, then some obstacles approached the robot in the consecutive images in a great speed with a too big difference so that the robot detected those obstacles too late (when it could only avoid the collision by doing a really sharp turn). When moving at lower speeds, this error did not occur, but then the robot moved too slowly in obstacle-free environments or when it should have moved in a simple straight line. For this reason, I defined a throttle control that is based on the distances from the sides and on

\[2\] The environment of the simulation is the corridor on the third floor in the main building of Obuda University, NIK faculty.
the distance in front (table I.). With this solution, the robot moves faster if there are no obstacles nearby and its speed is decreased as an obstacle approaches from some direction. I also applied this method on the original rule set, and this resulted in a control that oscillated much less. The details of the achieved results can be found in the Summary chapter.

B. Adaptive Field of View

Even when using the new throttle control algorithm, in narrow corridors the robot still moved too slowly, despite the fact that there were no obstacles in front of the robot on its path. The reason for this is that in the old version the speed control algorithm examined the obstacles on the sides independent from the current speed and orientation of the robot, and if these obstacles were too close, then the throttle controller’s output was never around the maximum value.

A possible solution for this is that if the robot moves at high speeds, then (similarly to the human perception) the obstacles on the sides will have less importance (or they will be ignored), because the chances that an obstacle on the side gets in the way of the robot is relatively small. (If we assume that the robot operates in a static environment, then the obstacles on the sides cannot move into the front of the vehicle.)

According to this, I limited the field of view of the robot based on the distances of the obstacles: I introduced an adaptively changed obstacle detection viewing angle (table II.). The essence of this is that if there are no obstacles in the front of the robot, then the two distances that were originally measured on the sides are changed to be measured almost in the front (producing a very narrow viewing angle, see figure 4(a)). As an obstacle approaches, the field of view gets wider and wider (figure 4(b)); basically this means that at high speeds the robot “zooms” to obstacles that are further away. Using this approach, the speed can be very well controlled (the distances on the sides are taken into account in the appropriate way) since obstacles on the sides are less considered when moving in a straight line at large speeds. If an obstacle approaches in the front, then the robot (as the field of view gets wider) can examine whether the obstacle continues towards the possible curve directions and it determines the angle of the turn according to this data.

The subsystem to determine the adaptive field of view has three Fuzzy inputs (left, right, front) and one Fuzzy output (the detection range). The maximum output value for the viewing angle was set to 31 degrees (to align with the field of view of the Kinect sensor, marked with green lines on figures 4 and 4(b)); while the area of the dead area is always marked as an obstacle [1]. The trapezoid membership functions of the output for the newly created controller can be seen in figure 5(b).

C. New Rules

Since the method originally described in [21] did not work always as expected (especially in regard for obstacles that are very close); a new rule system was developed using the old rules as its basis. This new method uses 16 upgraded rules (based on 16 rules fuzzy system described in [1]) to consider all different types of obstacles (the point-like obstacles in the front, the symmetrical walls on the two sides and the frontal wall-like obstacles as well, where the original method described in [21] had inadequate results [19]). The newly created rule system is the same as the rule set described in table III. if we skip the cells that have bold text within (6th row 3rd column, 7th row 4th column, 17th row, 18th row).

In the earlier version of the system ([19]) the first step was to define two Fuzzy inputs to determine the distances in the
front and in the diagonal directions. Since the primary goal is to move in a straight line, the distance from the obstacles in the front (figure 1.4(a), marked as \(l_F\) on figure 4), is more considered than the distances on the sides (figure 1.4(b), marked as \(l_L\) and \(l_R\) on figure 4). The number of the output membership functions was decreased to 5 opposed to the original version described in [21] (figure 1.4(c)); and as a result we got a suitably working navigational system that requires even less computational time [19]. The maximum output value for the degree of the steering wheel was set to 31 degrees to align with the maximum turn degree of the robot [1].

If we take a look at the non-bold rows (rows 2-4) of table III., it is visible that the new rule set directs the robot towards the centre line of the corridor. Rows 5-16 demonstrate the avoidance of the wall-like obstacles (also considering the walls that appear in the front, crossing the path of the robot). The 16th rule defines the avoidance maneuver of the suddenly appearing point-like obstacles in the front. It is visible that the robot always turns in the direction where there are no obstacles; and if this cannot be clearly determined, then it prefers the turn to the left. By comparing the original ([21]) rule set and the newly created controller we can determine that we managed to eliminate the biggest errors in the system that we used as a basis [1]. In addition to this, it can be seen from rows 8-16 of table III. that if there is an obstacle in front and very close to the robot, then it performs a sharp turn but it also considers the distances from the obstacles on the sides: it turns towards the side where the obstacles are further away. If the distances are the same, then it turns to the left.

By creating the rule set according to these, the system worked decently, but when going through longer narrow corridors, it approached the left-sided wall of the corridor too much; and then it avoided the collision with a sharp turn. To get rid of this error (apart from the introduction of the adaptive field of view); the extension and the modification of the original rule set of 16 rules was also necessary. One part of the cause of this error could be traced back to rows 6 and 7 of table III., because these conditions will be firing even if the walls on the sides are close, so they weaken the results of the already firing conditions in rows 3 and 4. As an effect, in narrow corridors if the moving direction of the robot is seen from an angle from the wall on the left, then the robot will move in a straight line until the distance from the wall is changed to a value in the “near” range. According to this, rules in rows 6 and 7 had to be modified so that they are not firing if the walls on the sides are close. The newly created rule set that contains 18 rules can be seen in table III. Rows 6 and 7 were modified according to the principles described above (see the bold text in rows 6 and 7 of table III.). The rules in rows 17 and 18 were also added so that they enforce the correct navigation in cases like this: as a result, if the walls on the sides are close, so they weaken the results 6 and 7 of table III., because these conditions will be firing even if the walls on the sides are close, so they weaken the results of the already firing conditions in rows 3 and 4. As an effect, in narrow corridors if the moving direction of the robot is seen from an angle from the wall on the left, then the robot tends to move to the right, and vice versa if the wall approaches from the right. The fuzzyfication of the output values is done using the already described membership functions that can be seen on figure 5.

By implementing the throttle control function into the new fuzzy deduction system with 18 rules, we get the path that can be seen in figure 6(c). If we compare this path with the path that the robot follows with the throttle control of the old system (figure 6(b).), it is clearly visible that the robot follows a more linear, more balanced path.

### V. Results

To determine the differences between the different navigational methods of the various rule systems, we used our simulator software. The simulator (by knowing the virtual environment and the parameters of the robot) determines the path of the robot which is also stored in an output file. The resulting images can be seen in figure 6, the basis of the simulation is the corridor on the third floor in the main building of Óbuda University, NIK faculty (we chose this
The firing conditions of the new Fuzzy controller with 18 rules. By leaving out the bold cells, we get the old rule set of 16 rules.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Frontal distance</th>
<th>Left-sided distance</th>
<th>Right-sided distance</th>
<th>Output angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Far</td>
<td></td>
<td></td>
<td>Zero</td>
</tr>
<tr>
<td>2</td>
<td>Not near</td>
<td>Near</td>
<td>Near</td>
<td>Zero</td>
</tr>
<tr>
<td>3</td>
<td>Not near</td>
<td>Near</td>
<td>Far</td>
<td>Positive</td>
</tr>
<tr>
<td>4</td>
<td>Not near</td>
<td>Far</td>
<td>Near</td>
<td>Negative</td>
</tr>
<tr>
<td>5</td>
<td>Medium</td>
<td>Far</td>
<td>Not near</td>
<td>Negative</td>
</tr>
<tr>
<td>6</td>
<td>Medium</td>
<td>Medium</td>
<td>Not near</td>
<td>Negative</td>
</tr>
<tr>
<td>7</td>
<td>Medium</td>
<td>Not near</td>
<td>Medium</td>
<td>Negative</td>
</tr>
<tr>
<td>8</td>
<td>Near</td>
<td>Near</td>
<td>Near</td>
<td>Very negative</td>
</tr>
<tr>
<td>9</td>
<td>Near</td>
<td>Near</td>
<td>Medium</td>
<td>Very positive</td>
</tr>
<tr>
<td>10</td>
<td>Near</td>
<td>Near</td>
<td>Far</td>
<td>Very positive</td>
</tr>
<tr>
<td>11</td>
<td>Near</td>
<td>Medium</td>
<td>Near</td>
<td>Very negative</td>
</tr>
<tr>
<td>12</td>
<td>Near</td>
<td>Medium</td>
<td>Medium</td>
<td>Very negative</td>
</tr>
<tr>
<td>13</td>
<td>Near</td>
<td>Medium</td>
<td>Far</td>
<td>Very positive</td>
</tr>
<tr>
<td>14</td>
<td>Near</td>
<td>Far</td>
<td>Near</td>
<td>Very negative</td>
</tr>
<tr>
<td>15</td>
<td>Near</td>
<td>Far</td>
<td>Far</td>
<td>Very negative</td>
</tr>
<tr>
<td>16</td>
<td>Near</td>
<td>Far</td>
<td>Far</td>
<td>Very negative</td>
</tr>
<tr>
<td>17</td>
<td>Medium</td>
<td>Near</td>
<td>Not near</td>
<td>Positive</td>
</tr>
<tr>
<td>18</td>
<td>Medium</td>
<td>Not near</td>
<td>Near</td>
<td>Negative</td>
</tr>
</tbody>
</table>

The comparison of the different methods is shown in table IV, where every row contains the average of four measurements. The values in the column “summed rotating angle” are the sums of the wheel angles measured once in every 500 milliseconds. The following column (“Max of summed rotating angles”) shows the maximum from the four measured sums. This is an important value, because due to the soft computing methods, the various routes are not the same, but the maximum rotation values are not that much different from the average value.

The resulting system provides a solution that follows a lot straighter path; and due to the adaptive speed control, it moves fast enough even in an unknown environment.

VI. SUMMARY

By examining the errors in the Fuzzy navigational system based on 16 rules that was already described in a previous article of ours, we developed a new navigational method that includes an adaptive control of the field of view for the sensors and a throttle control mechanism as well. The new obstacle avoidance system (that contains 18 rules) follows a much straighter path than the previous system with 16 rules, but due to the refresh rate of the map, at high speeds it was still possible that after the detection of the obstacle there wasn’t enough time for the robot to make a turn to avoid it. To prevent this from happening, a distance-based speed control was introduced, so that the vehicle slows down when it detects an obstacle near the robot. After introducing this feature, the robot moved too slowly when driving through narrow corridors, despite the fact that there were no obstacles in front of the vehicle. For this reason, I modified the angle of the measured distances on the sides so that this angle of measurement depends on the distances measured in the front (and also on the previously measured distances on the sides). This way the robot moves faster even in narrow corridors.
(a) Routes followed by the algorithm based on 16 rules [1] ((red, blue and green curves). shows the path and the speed of the robot extends an even with integrated throttle control and The starting point of the robot is in the circle the 16-rules-based method extended more balanced path than the one adaptive field of view moves fast in the bottom-right corner of the image. The with speed control, it follows a lot shown in figure 6(b).

(b) The curve with changing colours shows the path and the speed of the 16-rules-based method extended more balanced path than the previous system (shown with the blue curve). The time measurements that can be seen in table straighter path than the previous system (shown with the blue curve).

(c) The new system with 18 rules follows an even more balanced path than the one shown in figure 6(c). The resulting navigational system provides high speeds and follows a straighter path; and in addition to that, this method also increases the accuracy of the map building process, because the errors caused by the inaccuracy of the sensors and the actuators do not increase that fast as they did before in the old system with bigger and more frequent turns.

(d) The 18-rules-based system with integrated throttle control and adaptive field of view moves fast and follows a straight and balanced path (see table IV.). The improved version of the robot vehicle could be...
practically used for example to explore smaller areas after natural disasters, or to create maps automatically, or to protect critical infrastructures (for example as a patrol robot that circulates around the protected object).

VII. ACKNOWLEDGMENT

The author(s) gratefully acknowledge the grant provided by the project TÁMOP-4.2.1/B-11/2/KMR-2011-0001.

<table>
<thead>
<tr>
<th>Method</th>
<th>Traveled distance [m]</th>
<th>Summed rotating angle [deg]</th>
<th>Max of summed rotating angles</th>
<th>Lap time [sec]</th>
<th>Average speed [km/h]</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 Rules</td>
<td>56.6</td>
<td>1250.3</td>
<td>1707</td>
<td>127.8</td>
<td>1.6</td>
</tr>
<tr>
<td>16 Rules + speed control</td>
<td>55.6</td>
<td>1318.3</td>
<td>1499</td>
<td>239.5</td>
<td>0.8</td>
</tr>
<tr>
<td>16 Rules + speed + field of view control</td>
<td>55.9</td>
<td>573.3</td>
<td>696</td>
<td>180</td>
<td>1.1</td>
</tr>
<tr>
<td>18 Rules</td>
<td>56.2</td>
<td>817.0</td>
<td>1212</td>
<td>129</td>
<td>1.6</td>
</tr>
<tr>
<td>18 Rules + view angle control</td>
<td>56.2</td>
<td>1086.0</td>
<td>1862</td>
<td>129.3</td>
<td>1.6</td>
</tr>
<tr>
<td>18 Rules + speed control</td>
<td>55.7</td>
<td>764.0</td>
<td>785</td>
<td>226.5</td>
<td>0.9</td>
</tr>
<tr>
<td>18 Rules + speed + field of view control</td>
<td>55.6</td>
<td>569.3</td>
<td>595</td>
<td>179.5</td>
<td>1.1</td>
</tr>
</tbody>
</table>

TABLE IV

Comparison of the various navigational systems (averages of four measurements): one lap is displayed in figure 6(a) from the “start” point to the “lap time” line. The column “summed rotating angle” contains the sums of the wheel angles measured once in every 500 milliseconds.


