university of groningen / faculty of science and engineering

Robotics Practical 2 – WBAI030
Academic Year 2021-2022

Alexandru Dimofte (s4019199)

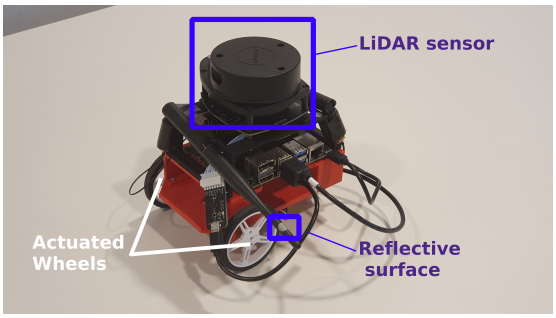
group07

Foreword: before proceeding with reading/grading the report, please note that without the figures and, of course, this paragraph, this document is around 4 pages and a half long. Moreover, admittedly, me and my lab partner Jakub Lucki have shared some of the figures we each created. When a figure was made by Jakub Lucki, it has been indicated as was appropriate, like it was the case with any other material taken from other sources. Apart from this, no other report-related information was shared between the two of us in the making of this report.

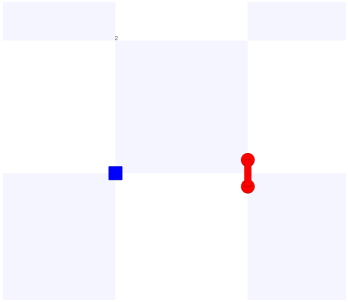
I. Simulation and Robot - [0.5 points]

The environments used were created in the Stage Simulator. Stage simulator is a 2(.5)D robot simulator written in C++, which can be used to simulate walls, robots and different sensors such as LiDAR, Odometers, etc. The main advantage lies in its simplicity, as one can draw a simple environment in a program of their choice (e.g. *Paint* on windows OS), save it as a PNG file and create a ROS launch file pointing to the picture, thus using it directly.

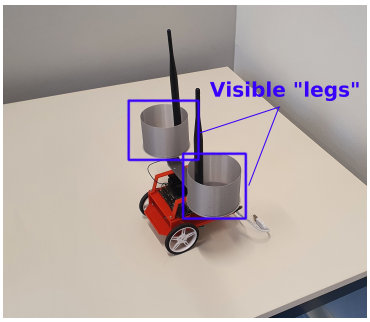
The robots used throughout this project were three unicycle mobile robots (figure 1a), with hardware and software designed by the Robotics Research Lab staff, Rik Timmers and Marc Groefsema (SOURCE: Robotics Practical 2 slides). Each robot has 2 wheels which provide 2 degrees of freedom, and a LiDAR mounted on top. In the real world, the positions of each robot was tracked by an OptiTrack motion capture system which used reflective dots placed on each robot in order to capture its movement (see reflective surface example in figure 1a). In the simulation a simulated odometer was used, as well as a simulated LiDAR. Furthermore, a robot was represented by a square (see figure 1b).



(a) Unicycle robot with a LiDAR sensor atop, two actuated wheels and several reflective dots used by the OptiTrack System to track the robot's movements



(b) Unicycle robot in simulation (blue square), and legs (two binded green circles) as viewed from atop



(c) Cylinders placed on top of another robot, which act as legs that a robot can follow

Fig. 1: Robot and Legs

Two vertical cylinders of roughly 5 cm diameter were bounded together at a distance of roughly 15 cm from eachother in order to represent *legs*. The robots had to follow the legs during some tasks as explained in further sections. Figure 1b shows the legs in the simulation (green circles binded together), and figure 1c shows them in real life.

II. Laser data clustering, Feature Extraction and Classifier - [3.5 points]

This section describes the process of classifying the legs as legs. In order to do this, a similar approach to the one described by [3] was used.

The first thing we did towards classifying legs was gathering data. In this project, like in [3], data was gathered using a LiDAR sensor mounted atop of each robot, which works by shooting lasers across 360° and waiting for their reflections to return. Using the time it took for a reflected laser to be returned back to the sensor, as well as the direction from which it returned, the location of a point hit by the laser is calculated in polar coordinates $(r,\ \theta)$. These are subsequently converted to cartesian coordinates $(x,\ y)$ following the equations below:

$$x = r * cos(\theta) \qquad\qquad y = r * sin(\theta)$$

The result is a set of points in cartesian coordinates which can be further used to estimate the position of different objects. To accomplish this, the same clustering algorithm used in [3] was employed in this project. The algorithm

works by calculating the euclidian distance in meters between each two consecutive points ($p_i \in R^2$) found by the LiDAR sensor as follows:

$$distance(p_1, p_2) = ||p_1 - p_2|| = \sqrt{(x_{p_1} - x_{p_2})^2 + (y_{p_1} - y_{p_2})^2} \qquad for\ any\ p_i = (x_{p_i},\ y_{p_i})$$

At any distance D higher than a chosen threshold distance $D_{threshold}$, a new cluster is defined and all previously found points are added to it. The process repeats until all the points are classified within some number of clusters N_c, which would ideally be the same number as there are objects in the visible range of the robots. To maximise the possibility of this ideal situation, we chose a threshold distance $D_{threshold}$ that reflects the closest distance that two objects can be relative to each-other. In the simulation, this number was found to be $D_{threshold} = 0.09$m. Remarkably, this value was the same as the one found in the real world.

For each cluster c, we used the points it contained to create three features: the girth $G(c)$, width $W(c)$ and depth $d_{depth}(c)$ of the respective cluster. In their work, [3] define a scenario in which m points belonging to one leg are collected and put into the same cluster. For each point p_i in this cluster, where $i \in [1, m]$, the girth is defined as the sum of all distances between each two consecutive points p_i and p_{i+1} (see **Girth** equation below) and the width is defined as the length of the line created by the first point p_1 and the last, p_m (see **Width** equation below). Finally, the depth is defined as the maximum distance between a point p_i and the line segment $\overline{p_1 p_m}$, and it can be calculated using the **Depth** equation below.

$$\textbf{Girth: } G(c) = \sum_{i=1}^{m-1} ||\overline{p_i p_{i+1}}|| \qquad \textbf{Width: } W(c) = ||\overline{p_1 p_m}||$$

$$\textbf{Depth: } d_{depth}(c) = \max_i \frac{(L_2^T L_2)(L_1^T L_1) - (L_1^T L_2)^2}{(L1^T L_1)} \qquad where\ L_1 = p_m - p_1 \quad \& \quad L_2 = p_i - p_1$$

Because the equations above require a cluster to have at least 3 points, all clusters with less than 3 points were filtered out before calculating the features.

The next step requires to train a Support Vector Machine (SVM) model to classify whether a cluster c depicts a leg or not, based on the features described above $G,\ W,\ d_{depth}$. SVMs are a family of algorithms that estimate hyperplanes used to split some given training data into different categories. If the data is n-dimensional, then the hyperplane is of dimension $n - 1$. In our project, there are only two categories to consider, leg or not leg, and therefore only one hyperplane is needed. Furthermore, the SVM uses three features to perform classification (the data is 3-dimensional), and hence the hyperplane calculated was a simple 2D plane. In our model, the data could not be easily separated by a plane on its own, and thus the radial basis function (RBF) kernel was used. RBF adds an extra feature to the already existing features, splitting the data into one more dimension. By using this extra dimension, along with the other features, the data could be linearly separated. The equation below describes the radial basis function, using the γ parameter used in this project.

$$K(\mathbf{x}, \mathbf{x}') = exp(-\gamma||\mathbf{x} - \mathbf{x}'||^2) \qquad where\ \gamma = 1/(3 * Variance_{features}) \tag{1}$$

Finally, the inverse regularization C parameter describes the degree to which misclassifications are allowed. A very high value leads to higher performance, but also increases the chance of over-fitting, while a small value can lead to a higher degree of generalization, but lower performance. Upon fine-tuning this value, we settled on $c = 1000$.

As for the data, entries consisting of the three features G, W, d_{depth} and one label (leg/not leg) were collected separately for legs and obstacles across two different environments. In order to collect legs data, one robot was ran in an empty simulation environment with nothing else present except for a pair of legs (see figure 1b). However, in the real world, since there cannot be an empty environment, collecting legs data was made possible by filtering out all LiDAR hit points found further than $1.5m$, while making sure no objects except the legs were present within this radius. In contrast to legs data, collecting data on objects and walls was done the same way for both simulation and real world. A robot was ran in an environment containing walls, and any kind of object as long as it was not a leg. The environment used was the one depicted in figure 3a, with the only difference being that the legs were not present.

To ensure a high diversity of examples in our data, we moved the legs around the robot while collecting legs data and the robot around the environment while collecting walls & objects data until a total of 20443 entries were collected for each datasets. This means 20443 data entries were collected for the legs and 20443 for the walls &

objects, for a total merged dataset of $20443 * 2 = 40886$ entries. This was done both for the simulation and for the real world environment, resulting in a simulation dataset and a real world dataset, both of size 40886. 20% of each dataset was used for testing, leaving 80% for training. Since this is a classification problem, and the data we used was well balanced, we used accuracy as our main metric.

Upon training on the simulation data, the testing accuracy was 92.6%. Training on the real world data resulted in a testing accuracy of 91.8%.

III. Null-space Control - [3 points]

Null-space-based (NSB) control is a robot control method introduced by [1] which can incorporate multiple tasks by combining two approaches known as competitive and cooperative behaviour coordination. Consider a task defined as

$$\sigma_i = f(\mathbf{q}) \textbf{ with the relation} \qquad \dot{\sigma}_i = f'(\mathbf{q}) = \mathbf{J}(\mathbf{q})\mathbf{v}_i \tag{2}$$

where $\mathbf{q}$ refers to the system configuration containing the location of the robot x and its orientation θ, $\mathbf{J}$ is the configuration-dependent task Jacobian matrix and $\mathbf{v}_i$ is the velocity command given by the ith, containing a linear velocity v (in m/s) and a rotational velocity ω (rad/s).

In order calculate the necessary velocity that fulfills a task with the least amount of error, the pseudo-inverse of the Jacobian from equation 2 (denoted $\mathbf{J}^\dagger$) is calculated and multiplied by the desired change in task function as follows:

$$\mathbf{v}_i(t) = \mathbf{J}^\dagger \dot{\sigma}_{i,d} = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T)^{-1}\dot{\sigma}_{i,d} \tag{3}$$

where $\mathbf{J}^\dagger$ is the pseudo inverse of the jacobian $\mathbf{J}$ and $\dot{\sigma}_{i,d}$ is the change in the ith task function based on the desired configuration $\mathbf{q}_d$ such that $\dot{\sigma}_{i,d} = \mathbf{J}(\mathbf{q}_d)\mathbf{v}_i$. Equation 3 can, however, result in drifts [1] and therefore, an extra term is added to mitigate this:

$$\mathbf{v}_i(t) = \mathbf{J}^\dagger \dot{\sigma}_{i,d} = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T)^{-1}(\dot{\sigma}_{i,d} + K_i(\sigma_{i,d} - \sigma_i(t))) \tag{4}$$

where $K_i > 0$ is a constant which was tuned for each respective task to achieve an optimal balance between responsiveness and smoothness.

One of the main advantages of NSB control is that each task velocity $\mathbf{v}_i$, with $i \in \{1, 2, 3\}$ in this project, can be calculated in parallel in their own respective direction. NSB control then combines the velocity commands in the order of their priorities such that a the contribution of a lower priority task is the respective velocity command calculated independently, projected onto the null space of the immediately higher priority task. Thus the final equation is the following:

$$\mathbf{v}_d(t) = \mathbf{v}_1 + \big(\mathbf{I} - \mathbf{J}_1^\dagger \mathbf{J}_1\big)\big[\mathbf{v}_2 + \big(\mathbf{I} - \mathbf{J}_2^\dagger \mathbf{J}_2\big)\mathbf{v}_3\big] \tag{5}$$

In this project, all three robots implemented three tasks. In the order of their priority, these tasks are: obstacle avoidance $\mathbf{v}_1$, follow-me $\mathbf{v}_2$, and formation control $\mathbf{v}_3$.

Obstacle avoidance is the task that preserves a robot's integrity by keeping the robot at a minimum distance of 0.4 meters from any obstacle. It is ranked as the highest priority task because if the robot is damaged by an obstacle, none of the other tasks can be performed. To begin performing this task, a robot is actively checking all hit-points found by the LiDAR. If the robot is approaching an obstacle or passes the desired distance (0.4 m), then the task is activated as explained further. Consider $p_0 = (x_0, y_0)$, the position of the nearest obstacle to the robot found by the LiDAR in the robot's frame of reference, which is situated at some distance $\sigma_1 = ||p - p_0||$ from the robot's position $p = (0, 0)$ (it is 0 for both x and y because this is the robot's position in its own frame of reference). This distance is smaller than the desired distance d_0 between the robot and the obstacle (i.e. $\sigma_1 < d_0$). The Jacobian aligned with the direction of the obstacle avoidance task is therefore:

$$\mathbf{J}_1 = \big[-x_0\sigma_1^{-1},\ \ -y_0\sigma_1^{-1}\big] \tag{6}$$

which represents the unit vector pointing in the direction of the task and it can subsequently be used to project the next task $\mathbf{v}_2$ onto the null space of $\mathbf{v}_1$. The velocity command $\mathbf{v}_1$ can be calculated starting from equation 4.

Keeping in mind that the position of the robot is $p = (0, 0)$, we reach the final equation:

$$\mathbf{v}_1(t) = \big[-x_0\sigma_1(t)^{-1},\ \ -y_0\sigma_1(t)^{-1}\big] K_1(d_0 - \sigma_1(t)) \tag{7}$$

where the value of K_1 was tuned to achieve an optimal balance between responsiveness of the robot and smoothness of the trajectory taken by the robot. In the simulation, we found $K_1 = 3$ to work best, while in the real world, $K_1 = 1.3$.

Follow-me: Once two clusters were classified as legs by the SVM and their mean positions $c_1 = (x_1, y_1)$ and $c_2 = (x_2, y_2)$ were at a distance within $[1.5,\ 2.5]$m of each-other, they were regarded as a *pair of legs*, with coordinates at the mean point between the two legs, $p_f = \big(mean(x_1, x_2),\ mean(y_1, y_2)\big) = (x_f, y_f)$. This task has the purpose of maintaining the robot closest to the legs at a distance of $0.5 + / - 0.1$ m. Additionally, p_f was stored such that if during the next iteration, two or more pairs of legs are found, the one closest to p_f is selected.

Computing the Jacobian $\mathbf{J}_2$ and velocity command $\mathbf{v}_2$ of this task is virtually the same mathematically as in the previous task, with the exception being that they are only computed for the robot closest to the pair of legs rather than for all robots. The robot R closest to the pair is further referred to as the *leader robot* or R_{leader}. In order to assess which robot was the leader, at a frequency of 5Hz, a supervisor class compared the distances between each robot to the pair of legs, $\sigma_{2,R} = ||p_R - p_f||$ where $R \in \{1, 2, 3\}$ represents one of the three robots as follows:

$$R_{leader} = \underset{R}{\operatorname{argmin}}\, \sigma_{2,R}$$

Once the leader robot was found, calculating the jacobian and velocity command was done using the equations below:

$$\mathbf{J}_2 = \big[-x_f\sigma_2(t)^{-1},\ \ -y_f\sigma_2(t)^{-1}\big] \tag{8}$$

$$\mathbf{v}_2(t) = \big[-x_f\sigma_2(t)^{-1},\ \ -y_f\sigma_2(t)^{-1}\big] K_2(d_f - \sigma_2(t)) \tag{9}$$

such that $(x_f,\ y_f)$ became the position of the legs in the frame of reference of the leader robot, and for all other non-leader robots, $x_f = y_f = 0$, resulting in a null jacobian and a null velocity command. This assured that only the leader robot actively followed the pair of legs. Like in the previous task, K_2 was found to work best at $K_2 = 5$ in the simulation and $K_2 = 2.5$ in the real world environment.

Formation Control is a task introduced into NSB control by [2] with the aim of maintaining some number of robots (3 robots in this project) in a predefined formation based around a centroid ${}^0p_c = ({}^0x_c, {}^0y_c)$ where ^{0} indicates that a point is in the global frame of reference. The centroid was calculated as the mean coordinate of the three robots like shown in the equation below this paragraph. The structure of the formation defined for this project is displayed in figure 2.

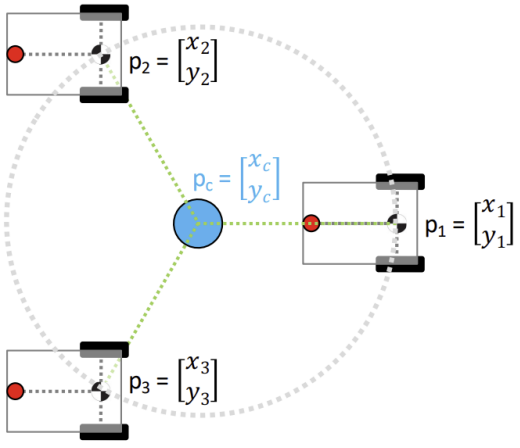


Fig. 2: Predefined formation imposed on three unicycle robots at positions $p_i = [x_i,\ y_i]^T,\ i \in \{1, 2, 3\}$ for a centroid position $p_c = [x_c,\ y_c]^T$. This formation is imposed by the third priority task in this project's NSB control - formation control (SOURCE: Robotics Practical 2 slides)

Mathematically, this triangular formation around the centroid represents the desired positions of each robot in

their own frame of reference:

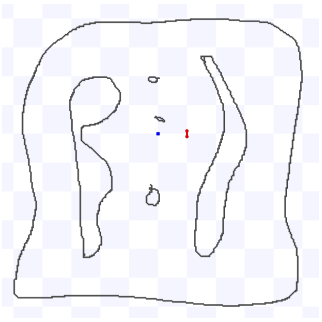
$$ {}^{0}p_c = 3^{-1}\sum_{i=1}^{3} {}^{0}p_i \qquad \sigma_{3,d} = \begin{bmatrix} {}^{1}x_c + d & {}^{1}y_c \\ {}^{2}x_c + d\ cos(120°) & {}^{2}y_c + d\ sin(120°) \\ {}^{3}x_c + d\ cos(120°) & {}^{3}y_c - d\ sin(120°) \end{bmatrix} \tag{10} $$

such that $d = 0.5$m and the coordinates of the centroid ${}^{i}x_c$, ${}^{i}y_c$ at row i are in the frames of reference of robot i with $i \in \{1, 2, 3\}$. The Jacobian of this task $\mathbf{J}_3$ is simply the 3D identity matrix, because there is no 4th task to be considered. For each robot i, the velocity command $\mathbf{v}_i$ is simply row i in matrix $\sigma_{3,d}$ which is subsequently multiplied by K_3. The final equation is $\sigma_{3,d}[i]K_3$ for each ith robot. Upon tuning, $K_3 = 4$ was found to work best in the simulation and $K_3 = 1.75$ in the real world environment.

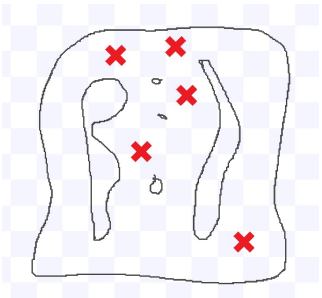
Putting the velocity commands and the Jacobians together using equation 5 results in one final velocity command $\mathbf{v}_d(t) = (v,\ \omega)$, which is further modified in two steps. First the velocities were limited within the intervals $v \in [-0.2,\ 0.2]$ m/s and $\omega \in [-2,\ 2]$ rad/s and second, ω was scaled by 10 (i.e. $\omega \leftarrow 10\,\omega$), thus constituting the low-level controller.

IV. Experiments and Results - [3 points]

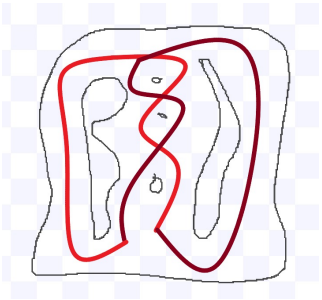
1. SVM experiments: Two experiments were designed in order to test a trained SVM model. They aimed to verify whether the model could reliably make the difference between a leg and an obstacle or wall. The first experiment consisted of a robot placed in an empty environment with only a pair of legs present (figure 1b above). In the real world environment, this was simulated by limiting the range of the LiDAR sensor to 1.5m and adding no obstacles within this radius. Since the coordinates of all the legs detected were marked, we could verify which legs are detected on `rviz`. The test was passed if the legs were consistently classified as legs. The second experiment consisted of using a robot in an environment with walls, obstacles and legs (figure 3a) to see whether the model detects the correct legs. The obstacles in the real world consisted of a box, a bucket and two towers (see figure 3d). Their locations were changed multiple times throughout the testing. Again checking `rviz`, we drove the robot around remotely while allowing small errors where some objects were recognised as legs in some angles because these would be accounted for when the robot checks for pairs of legs. During the simulation, we placed the robot at various different spots on the map (as shown by the red X's in figure 3b) where we tested the consistancy of the SVM. Subsequently, the robot was driven around the map following paths approximate to the ones shown in figure 3c. The test was considered passed if the SVM was consistently correct in its predictions. If any of the two tests were failed, more data was collected with more emphasis on its quality, and the SVM was retrained. This process was repeated until both experiments were passed.



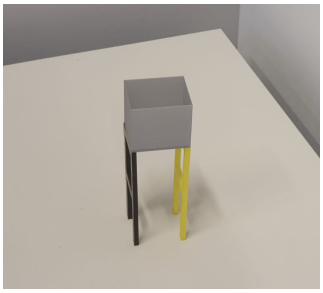
(a) Standard environment with one robot and legs



(b) Standard environment with red X's marking the spots at which a robot was placed in order to test the SVM on the surroundings. (SOURCE: Jakub Lucki)



(c) Approximate paths on which we drove the robot in order to test the SVM and NSB control tasks. (SOURCE: Jakub Lucki)



(d) Tower used as obstacle in the real world environment.

Fig. 3: Standard environment, spots/paths of testing and tower obstacle

NSB control experiments: These experiments were used both to test the system and to adjust the $K_i,\ i \in \{1, 2, 3\}$ parameters. The first set of experiments tested each of the three tasks in parallel, with two more experiments testing overall control.

To test obstacle avoidance in the real world, an obstacle was placed next to an object and moved slowly towards it. If the robot stayed away from the object at a consistent distance of 0.5m, then more objects were moved towards

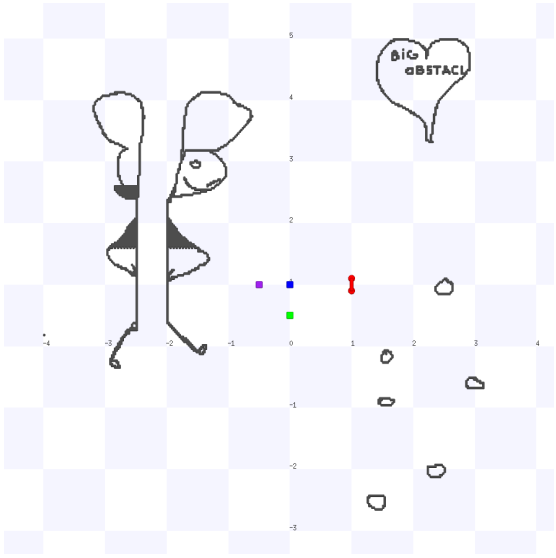
the robot in an effort to overwhelm it. If the robot managed to successfully avoid all the objects, as long as it had a reasonable amount of space to escape them, then the robot had passed the test. In the simulation, no dynamic way of moving objects exist and therefore, the walls were moved towards the robot. If the robot successfully avoided the walls, as long as they were not moved at a speed that the robot could not escape, then it passed the test. Throughout this process, multiple K_1 values were tried until the tests were passed.

For the follow-me task, we remotely moved the pair of legs around, while actively checking the distance between the robot and the legs. Additionally, obstacles such as towers (see figure 3d above) were placed in the real world, as a means to confuse the robot. For the simulation, two example paths taken by the legs and the robot can be seen in figure 3c above. If the robot consistently stood close to the pair of legs at around 0.5m distance, then it passed the test. Similarly as above, multiple K_2 values were tried for both environments until the robot passed the test.

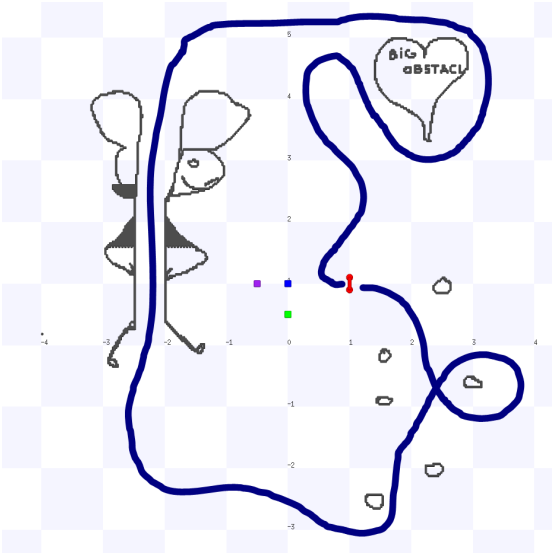
The formation control task was tested first by placing the robots in an empty environment (i.e. no obstacles or legs in their vicinity) in order to let the robots arrange in a triangle formation. The starting points of the robots were changed multiple times in order to test multiple scenarios. Once the robots successfully moved in the formation, we restarted the program, allowing robot 1 to be remotely controlled. Robot 1 was first moved around the environment to see if the other two robots would follow in formation and then it was intentionally steered into the other robots in an effort to break the formation. If the formation stayed consistent throughout the entire process, the test was deemed passed. Again, multiple K_3 values were tried until the system functioned as desired.

Once all the tests thus far were passed, the overall control was tested. The first experiment was aimed at checking if the follow-me task worked harmoniously with formation control. Therefore, this was tested in an empty environment with three robots and a pair of legs. For this test to be passed, the robot closest to the legs had to follow the legs directly, while the other two should follow their desired positions within the formation. During this experiment, the legs were moved at different times away from the robots, and back towards them in order to test their evasive reactions. If the behaviour was deemed acceptable, the second experiment tested whether the swarm of robots could successfully follow a pair of legs through an environment filled with obstacles. We did this in the real world by placing all the obstacles available to us at random locations throughout the lab and remotely controlling the pair of legs. In the simulation, this was tested on different paths on the standard environment (see figure 3a for standard environment and figure 3c for examples of paths) as well as on paths through a custom made environment made by us (see figure 4a for the custom environment and figure 4b for a possible path taken through the testing of overall control in the simulation). In both simulation and real world, the robots successfully followed the legs, while also maintaining the formation whenever it was possible.

Having passed all the tests, we deemed the overall NSB control to work as intended.



(a) Custom made environment with individual small obstacles, one big obstacle, and a tight passage.



(b) A possible path taken through the testing of overall control.

Fig. 4: Custom environment and an approximate path taken through it while testing NSB overall control.

REFERENCES

[1] G. Antonelli, F. Arrichiello, and S. Chiaverini. The null-space-based behavioral control for autonomous robotic systems. *Intelligent Service Robotics*, 1(1):27–39, 2008.

[2] G. Antonelli, F. Arrichiello, and S. Chiaverini. Experiments of formation control with multirobot systems using the null-space-based behavioral control. *IEEE Transactions on Control Systems Technology*, 17(5):1173–1182, 2009.

[3] W. Chung, H. Kim, Y. Yoo, C.-B. Moon, and J. Park. The detection and following of human legs through inductive approaches for a mobile robot with a single laser range finder. *IEEE transactions on industrial electronics*, 59(8):3156–3166, 2011.