# EXPERIMENTAL TESTBED FOR AUTONOMOUS FLIGHT WITH NANO QUADCOPTERS

# Contents

# 1. Overall view of the project

This section gives an overall view of the project.

## 1.1. System information

Computer: Microsoft Surface Pro. Model NO: 1866
Computer OS: Dualboot Ubuntu 20.04.2 Desktop LTS
Secure boot password: nebolab1
user: nebolab
password: nebolab

## 1.2. Dictionary

Crazyflie (CF) = Bitcraze's nano quadcopter. Our model was Crazyflie 2.1

LPS = Loco position system

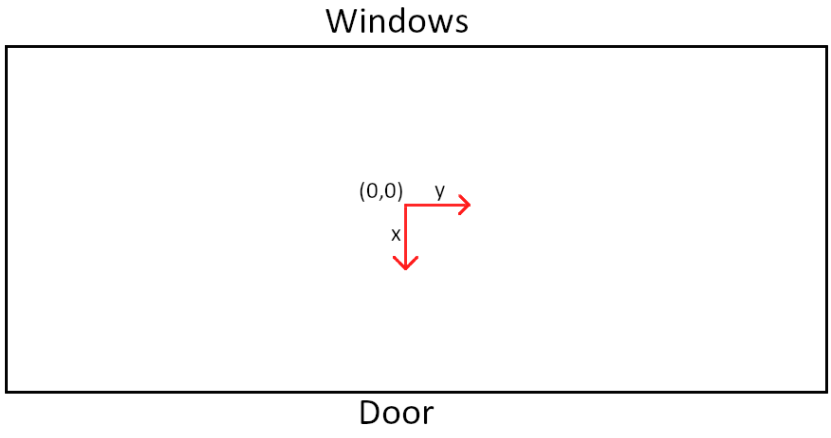
VM = Virtual machine

VB = Oracle VM Virtual Box

PFPP = potential field path planning

## 1.3. Notes

- Female pin headers of the decks are very sensitive to broke, so be very gentle when detaching decks from the pins. One loco deck female header broke when removing deck from the CF. Although female pin headers are spare parts and now the lab has a few of them, so it is possible to solder a new header to the deck.
- It is very important that the positive x-axis and Crazyflie's front are both heading in the same direction. I was suffering huge balance problems because I tried to take off while Crazyflie was heading to the negative x-axis. The positive x-axis is heading towards the lab's door, as presented in Picture 1.




Picture 1: Axis directions and the origin.

- The origin of the 3d coordinate space is the middle of the floor. 0 height (z-coordinate) is level of the lower anchors.

## 1.4. The current state of the project

Loco position system installed and running, all lab's decks are tested except for LED-ring, autonomous flight examples with both individual CF and multiple CF's, some accuracy

measurements has been done, potential field path planning algorithms are tested with two CFs, Crazyswarm and ROS are installed and configured.

# 2. Setting up environment

This section contains instructions on how to set up the environment.

## 2.1. Virtual machine

This section includes instructions to set up Bitcraze virtual machine running over Windows and how to connect quadcopter to that. Using VM was a great way to get started, but virtual machine causes around 40ms latency https://github.com/bitcraze/crazyflie-lib-python/blob/master/sys_test/single_cf_grounded/test_link.py, so I ended up running client locally. But if you want easy to get your first touches to the client and crazyflie, it could be a good idea to begin with VM at least. There were around 4ms of latency when running the client locally. If you want to install the client locally, you can skip this section.

This setup is tested with Oracle VM virtual machine, which can be downloaded from https://www.virtualbox.org/wiki/Downloads.

Installation instructions are available here: https://www.bitcraze.io/documentation/tutorials/getting-started-with-crazyflie-2-x/#inst-virtualmachine

To enable USB 2.0 and USB 3.0, the Virtual Box Extension Pack is required. The pack is available from the same address as VB.

One common error is “VT-x is disabled in the BIOS for all CPU modes”. To solve the problem, you need to go to your physical computer’s BIOS and enable virtualization from there.

Configuring USB instructions are available here: https://www.bitcraze.io/documentation/tutorials/getting-started-with-crazyflie-2-x/#config-usb-vm.

## 2.2. Setting up the local client (Ubuntu 20.04.2)

Bitcraze documentation provides great instructions to set up the Python client to the Linux https://www.bitcraze.io/documentation/repository/crazyflie-clients-python/master/installation/install/. I had some troubles with connecting Crazyradio to the client. Problems were caused by missing udev permission rules. Instructions on how to add udev permissions can be found on GitHub https://github.com/bitcraze/crazyflie-lib-python/blob/master/docs/installation/install.md.

## 2.3. Crazyflie

Bitcraze offers great instructions https://www.bitcraze.io/documentation/tutorials/getting-started-with-crazyflie-2-x/ to assemble the Crazyflie. After assembly, the firmware must be updated. Updating crazyflie follow steps presented by Bitcraze: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/building-and-flashing/build/

After that master branch of the cflib is also needed to get and install. To do that, follow the instructions provided by the Bitcraze forum: https://forum.bitcraze.io/viewtopic.php?t=4820.

## 2.4. Localization system

Following these instructions https://www.bitcraze.io/documentation/tutorials/getting-started-with-loco-positioning-system/, the loco positioning system can be set up. To get overall understanding how LPS works, here is more information: https://www.bitcraze.io/documentation/system/positioning/loco-positioning-system/.

## 2.5. Cflib

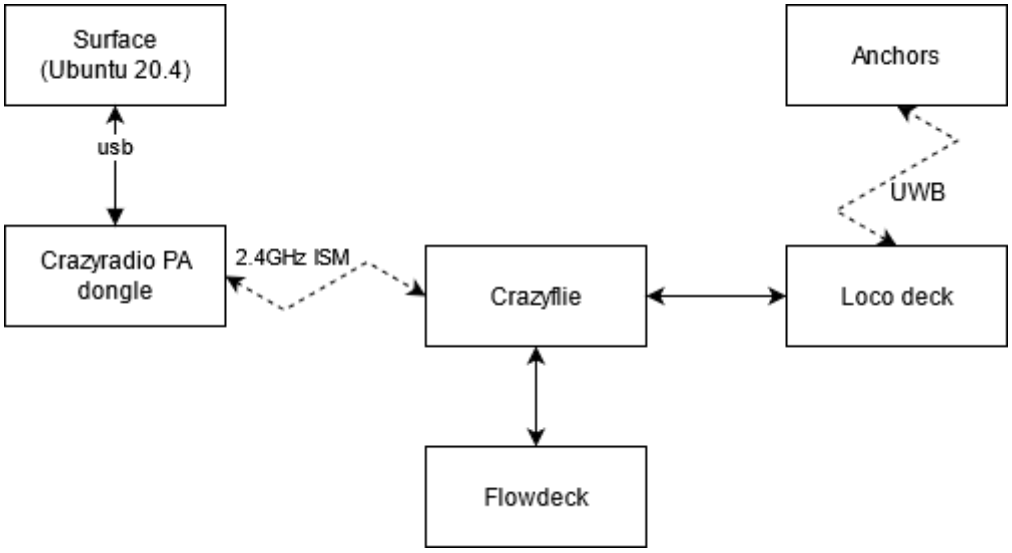
Cflib enables API, which communicate between Crazyflies. Documentation is available here: https://www.bitcraze.io/documentation/repository/crazyflie-lib-python/master/. Cflib provides couple of Python written examples, which enables entry level testing with different decks and autonomous flight for example.

## 2.6. GitLab

GitLab has a repository for project related code https://gitlab.tuni.fi/kdanay/comaswa/-/tree/crazyflie/Crazyflie. Crazyflie branch includes two directories: matlab and potentialFieldPathPlanning.

### 2.6.1. High level description of the hardware

GitLab codes are tested with the hardware showed in Picture 2.



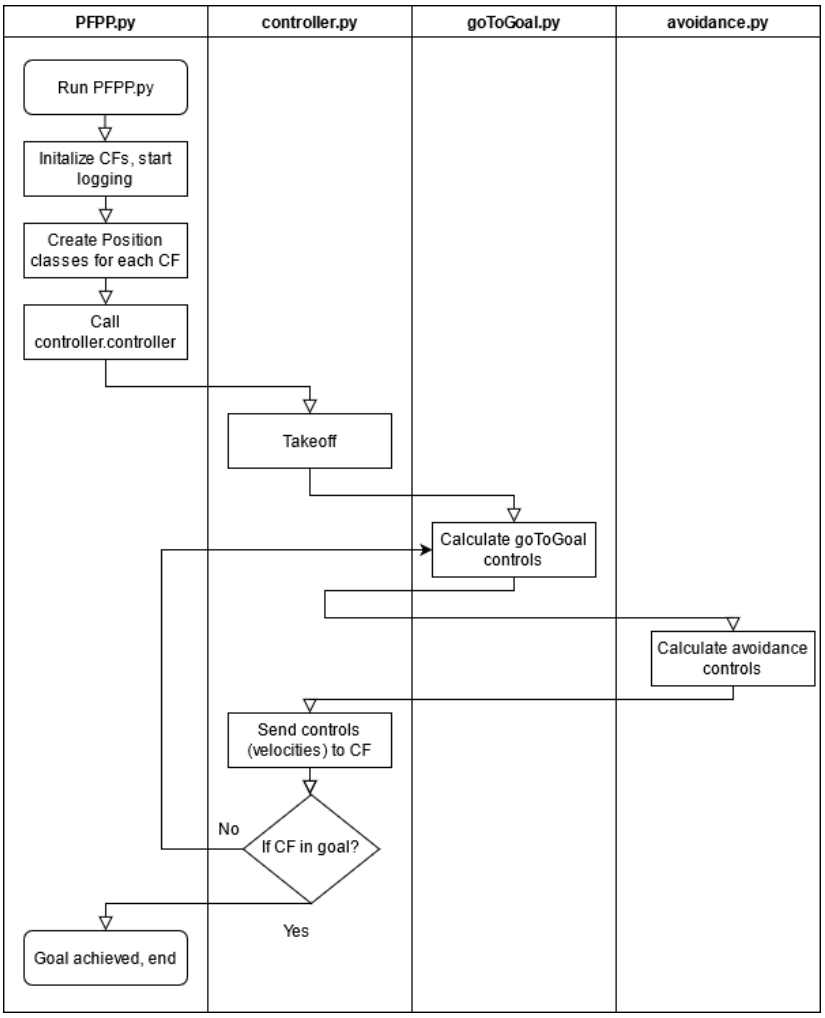

Picture 2: Used hardware and their communications.

Python scripts are executed in Surface. With help of cflib library, communication with Crazyflies is possible via Crazyradio PA USB dongle. Crazyflie and Crazyradio are using 2.4GHz ISM. To enable 3d-positioning Loco deck is mounted to the Crazyflie. Loco decks are communicating with lab's eight anchors via UWB, and 3d-position is calculated onboard. Flowdeck enables accurate z-axis coordinate.

### 2.6.2. Directories

Matlab directory includes Matlab and Simulink models of the PFPP. PFS.m is script that runs pFSimulator.slx Simulink model. PFSimulator.slx simulates the PFPP algorithm and CF route. In simulation CF dynamics are simplified as single integrators, and their controls (inputs) are velocities and output are positions. PFS.m makes plotting that shows the simulated route of the CF and also real route if position log is available. More information of simulations can be found from section 5 of this document.

PotentialFieldPathPlanning directory includes needed scripts to flight autonomously avoiding obstacles. Picture 3 above illustrates how PFPP works.




Picture 3: Swimlane diagram of PFPP structure.

PFPP.py initializes CF and start logging position information. After that Position is created for each CF and controller.controller is called. In controller CF defines goal coordinates and took off. Then goToGoal is called and its controls is calculated. After that avoidance controls are calculated in avoidance.py and controller.py sends controls to the CF. Finally, if CF is in its goal, goal is achieved and script ends. Otherwise, new loop begins with calculating goToGoal controls.

### 2.6.3. Running instructions

With these instructions you can test the PFPP with Crazyflies with loco deck (and flow deck to more precise z-axis accuracy).

- Clone the repository to your computer.
- Navigate to the potentialFieldPathPlanning directory.
- Attach Crazyradio Dongle to the computer.
- Check from PFPP.py which CFs are hardcoded to the file and power them on.
- Put Crazyflie to the floor. In two CF example I recommend using coordinates approximately (-2,0) and (2,0). There are no restrictions where CFs should be placed, but it is recommended to compare starting points to goal points and simulate the test case with Matlab first.
- Execute “python3 PFPP.py”.

I used mainly PyCharm Community Edition 2021.1.3 to running and to modifying the scripts.

After this position logger file(s) are saved to the same directory. If you want to plot real positions with Matlab code PFS.m, insert path to those pos files to the PFS.m script.

NOTE: In two CF avoidance code a crashing bug has appeared few times. In the bug CF position information is way too delayed and although CFs are bypassed each other's and they are quite close to their goals, position information says that they are almost in starting points still. That causes CF just going straight until code is stopped or CF has crashed.

### 2.7. Crazyswarm & ROS

Crazyswarm's documentation is available here: https://crazyswarm.readthedocs.io/en/latest/index.html. According to them, Crazyswarm is a "Platform that allows you to fly a swarm of Bitcraze Crazyflie 2.x quadcopters in tight, synchronized formations.".

#### 2.7.1. Installation

Crazyswarm installation follows these steps: https://crazyswarm.readthedocs.io/en/latest/installation.html. I installed option "Physical robots and simulation". The surface has ubuntu 20.04 and ROS version is Noetic.

#### 2.7.2. Configuration

Configuration instructions are here: https://crazyswarm.readthedocs.io/en/latest/configuration.html. Crazyflies address form is following `0xE7E7E7E70X`. The last number (X) is marked onboard with paper and tape.

Crazyswarm required firmware is installed to following Crazyflies:
`0xE7E7E7E70X`
`0xE7E7E7E70X`

Crazyswarm required radio firmware is installed to radio with a red marker.

I followed "unique marker arrangements" configuration instructions but set "motion_capture_type:none" because, at the moment motion capture is not used with this setup. LPS is used as a localization system.

I ensured that installation and configuration succeed with this example: https://crazyswarm.readthedocs.io/en/latest/tutorials/tutorials.html. Remember to execute "source devel/setup.bash" before running Python script.

## 3. Decks

This section describes all tested decks. There are some limitations which decks can be used at the same time. Combability matrixes are available here: https://www.bitcraze.io/documentation/system/platform/cf2-expansiondecks/#compatibility-matrixes.

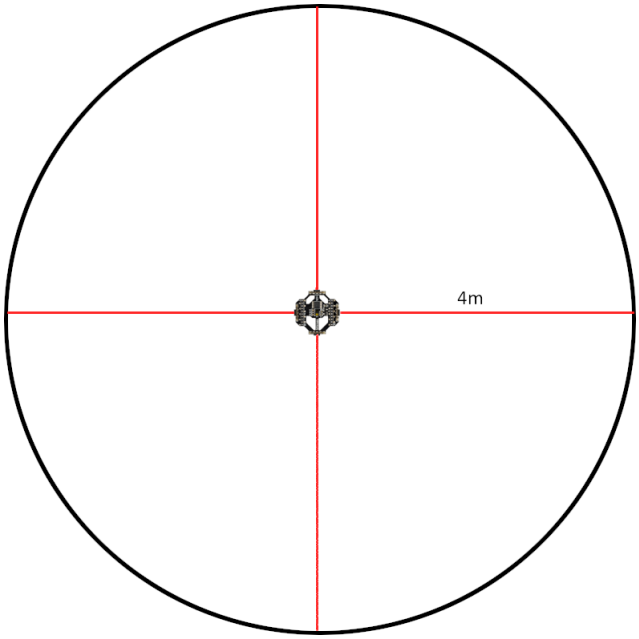
### 3.1. Loco deck

Loco deck is used with LPS, and its documentation is here: https://store.bitcraze.io/collections/decks/products/loco-positioning-deck.

## 3.2. Multi-ranger deck

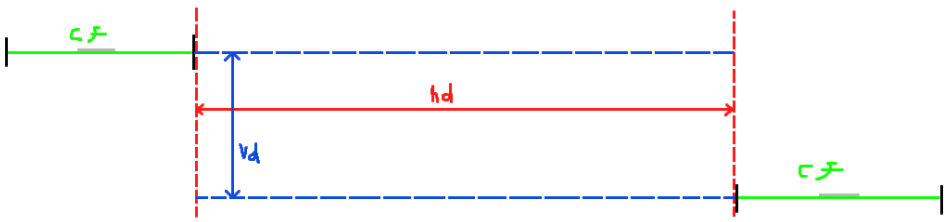
Multi-ranger deck documentation is available here:
https://store.bitcraze.io/collections/decks/products/multi-ranger-deck.

Multi-ranger deck has five high accuracy lasers, which can detect objects up to 4 meters. Picture 4 presents multi-ranger detection directions, expect for the laser above the multi-ranger deck.




Picture 4: Presentation of multi-ranger deck detection directions, expect for the laser above.

I investigated the multi-ranger deck's ability to detect other Crazyflies. I created test code and compared how the vertical and horizontal distances affect to MR deck's ability to detect obstacles. Distances are shown in picture 5.




Picture 5: Distances between two Crazyflies

In the first test, vertical distance (vd) was 0.0m, and Crazyflies were 0.3m above the ground. Crazyflies detected each other all tested horizontal distances (0.5m, 0.7m and 1.0m). In the second

test, the vertical distance was changed to 0.05m. Crazyflies were still able to detect each other all tested horizontal distances (0.5m, 0.7m, 1.0m). Finally, the vertical distance was set to 0.10m. In this case, Crazyflies did not detect each other anymore if the horizontal distance was 0.5m or 0.7m. And when hd was set to 1.0m, detection worked only a few times.

### 3.3. Flow deck

Flow deck's documentation is available here: https://store.bitcraze.io/collections/decks/products/flow-deck-v2. Flow deck provides z-axis accuracy very precisely compared to LPS z-axis accuracy. If only LPS is used, z-axis accuracy can be quite poor, as described in section 4.

### 3.4. AI deck

While writing this documentation, AI-deck was still in early access mode, which means that hardware was ready and tested, but the software was working in the process. AI-deck's info page: https://store.bitcraze.io/products/ai-deck-1-1?variant=32485907890263.

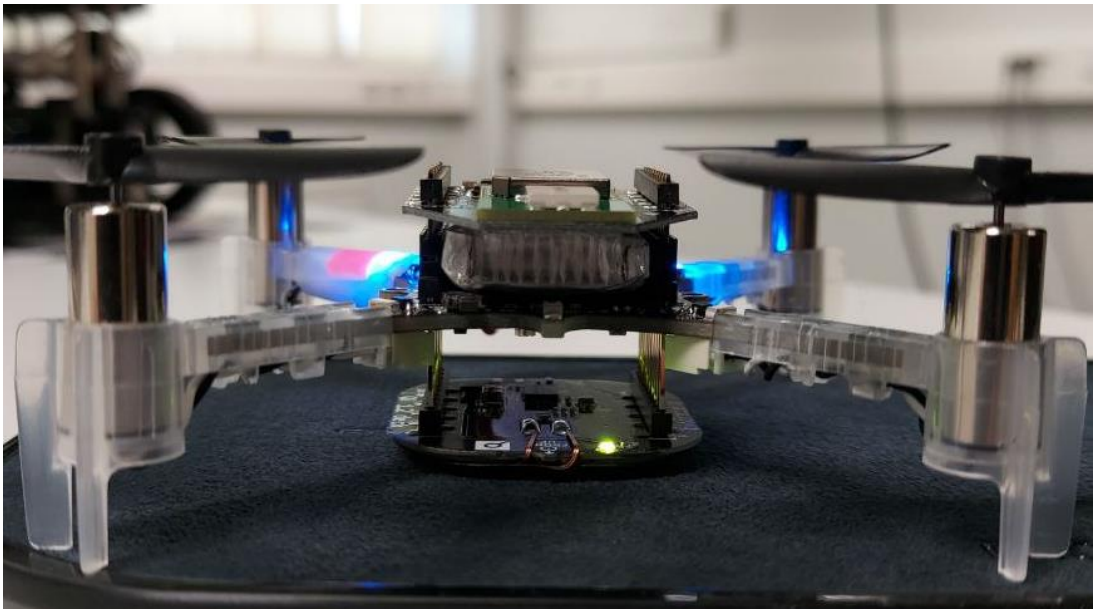
At the moment, AI-deck's use cases are connected to AI applications. One example includes streaming images from the AI-deck to the computer screen. The example code is available here: https://github.com/bitcraze/AIdeck_examples/blob/master/NINA/viewer.py. I tried that example, but there was not needed to investigate camera-related applications further, so that was my only test with the AI-deck.

In our case, AI-deck's calculating capacity and WIFI module were interesting. Could there be any reasonable way to utilize AI-deck to building distributed system? I asked the question from Bitcraze, and the answer is here: https://forum.bitcraze.io/viewtopic.php?f=21&t=4996.

If there is a need to develop AI-deck applications in the future, the following blog post and workshop give a good overall understanding of AI-deck: https://www.bitcraze.io/2021/05/starting-development-with-the-ai-deck/.

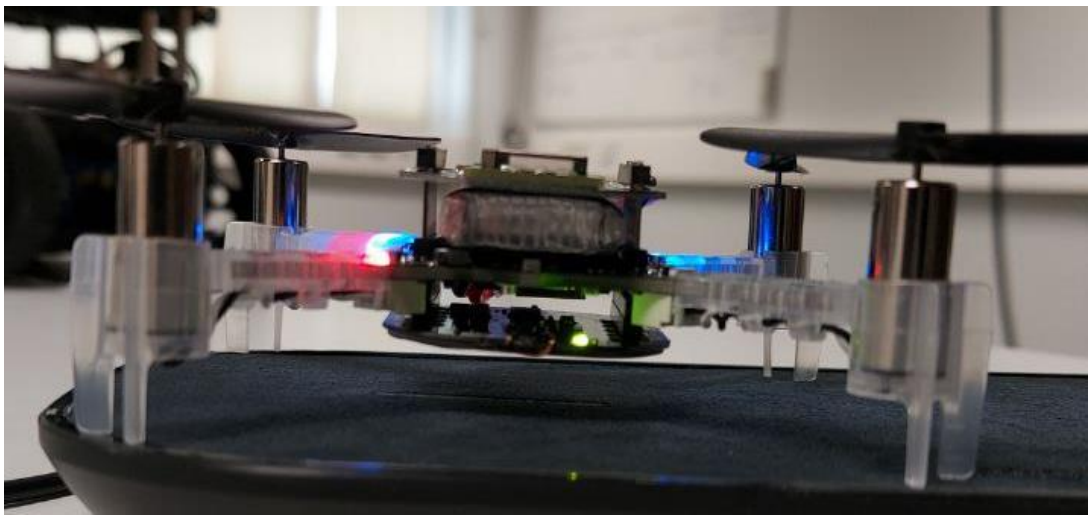
### 3.5. Charger deck

The charger deck's documentation and technical details are available here: https://www.bitcraze.io/products/qi-1_2-charger-deck/. I tested the charging deck with two different pins. Long pins, which comes in CF start package, and extra-long pins, which comes with AI-deck. With extra-long pins (AI-deck's pins) charging connection was stable because the deck almost touched the charging pad. CF was also allowed to move slightly from the center (around 1cm). Picture 6 shows the case with extra-long pins.



Picture 6: Charger deck with extra-long pins.

With long pins (picture 7) wireless charging was still possible, but the connection was quite unstable. CF had to be set to the pad very precisely because if CF wasn't in the middle of the pad, charging didn't work.



Picture 7: Charger deck with long pins.

So, it is possible to charge with long pins, but it is too unstable to use. At least if the goal is to use a charger deck so that CF would land to the pad autonomously. With extra-long pins and with a more accurate position system, I think that it would be possible. At the moment positioning system is LPS, which accuracy is not enough to test the charging deck while flying. Especially z-axis accuracy is relatively poor without Flowdeck, and it is not possible to use Flowdeck and charger deck at the same time.

# 4. Tests

## 4.1. LPS accuracy measurements

Crazyflie provides 10cm accuracy if the Loco position system is installed correctly. I run a couple of tests to investigate the accuracy of LPS in three different modes. Modes were TWR, TDoA2, and TDoA3. TDoA3 should be the most precise one. More documentation of different modes is available here: https://www.bitcraze.io/documentation/repository/lps-node-firmware/master/protocols/twr-protocol/.

All measurements were taken in Nebolab. Measurement values were logged with a little bit modified basiclog script https://github.com/bitcraze/crazyflie-lib-python/blob/master/examples/basiclog.py. The code took x,y, and z values of stateEstimate once per second. 20x3 values were taken with each measurement, and values were written to the CSV file. Same Crazyflie was used in every measurement.

### 4.1.1. Terms

**Estimated position**: Position, which Crazyflie calculates based on the information from the anchors. Combination of x, y, and z-coordinate.

**Real position**: The real psychical position where Crazyflie is. This position is measured manually with a tape measure from the anchor chip to the Crazyflie chip. Combination of x, y, and z-coordinate.

### 4.1.2. Stationary accuracy

This measurement goal is to investigate what is the standard deviation of Crazyflie's estimated position and what is the difference between the estimation position and the real position. Measurements are made for each mode, and after that differences between modes can be compared.

The measurement event occurred as follows:

1. Crazyflie was set to the reference position (real position: 2.16, 2.67, -0.31)
2. poslog.py script was runned and 20x3 estimated coordinate values was logged into csv file.
3. The estimated position was calculated as follows: 20 measured values were added together, and the result was divided by 20. This was repeated to all estimated coordinates (x, y, z).
4. The variance was calculated as follows: LibreOffice VAR-function calculated the variance of 20 measured values. This was repeated to all estimated coordinates (x, y, z).
5. Standard deviation was calculated as follows: The square root was taken from the result of step 4.
6. Crazyflie was taken off by hand and moved around the lab.
7. Crazyflie was set to the reference position (real position: 2.16, 2.67, -0.31)

The measurement above were repeated five times for each mode.

After that, figure 1 results were calculated.

| TWR | x | y | z |
|---|---|---|---|
| variance avg | 1.83E-04 | 1.42E-04 | 2.02E-05 |
| stan dev avg | 1.31E-02 | 1.19E-02 | 4.45E-03 |
| ref point | 2.16 | 2.67 | -0.31 |
| est position avg | 2.25 | 2.77 | 0.12 |
| error | 0.09 | 0.10 | 0.43 |

| | | | |
|---|---|---|---|
| | | | |
| TDoA2 | x | y | z |
| variance avg | 2.73E-04 | 2.18E-04 | 1.11E-03 |
| stan dev avg | 1.63E-02 | 1.42E-02 | 3.09E-02 |
| ref point | 2.16 | 2.67 | -0.31 |
| est position avg | 2.34 | 2.71 | -0.23 |
| error | 0.18 | 0.04 | 0.08 |
| | | | |
| TDoA3 | x | y | z |
| variance avg | 3.85E-04 | 1.97E-04 | 6.20E-04 |
| stan dev avg | 1.93E-02 | 1.39E-02 | 2.45E-02 |
| ref point | 2.16 | 2.67 | -0.31 |
| est position avg | 2.30 | 2.74 | -0.27 |
| error | 0.14 | 0.07 | 0.04 |

Figure 1: Stationary measurements results.

Variance avg is the average of variances calculated in step 4. Stan dev avg is the average of standard deviations calculated in step 5. The reference point is the real position of Crazyflie, which is measured with a tape measure. Est position avg is the average of estimated positions calculated in step 3. Error is the difference between est position avg and ref point.

### 4.1.3. Distance accuracy

This measurement goal is to find out how accurate Crazyflie's estimated position is when moving crazyflie from real position: 2.16, 2.67, -0.31 to some other position along one axis at a time. In this measurement, it is irrelevant how much error there is between the estimated position and the real position. The only thing which is being investigated is the following: Is the distance between two estimated positions the same as the real distance that Crazyflie was moved.

The measurement event occurred as follows:

1. Crazyflie was set to the marked position (real position: 2.16, 2.67, -0.31)
2. poslog.py script was run, and 20x3 estimated coordinate values were logged into a CSV file.
3. The estimated position was calculated as follows: 20 measured values were added together, and the result was divided by 20. This was repeated to all estimated coordinates (x, y, z).
4. Crazyflie was moved along one axis at a time. First time +1.20m y-axis.
5. poslog.py script was run, and 20x3 estimated coordinate values were logged into a CSV file.
6. The estimated position was calculated as follows: 20 measured values were added together, and the result was divided by 20. This was repeated to all estimated coordinates (x, y, z).
7. The mode was changed, and this procedure was repeated until all modes were measured.

After gathering measurements of the y-axis, the same procedure was repeated with the x-axis (-1.20m) and lastly with the z-axis (+0.64m). After that, Figure 2 results were calculated. Figure 2 results were calculated as follows:

1. Calculate the distance between two estimated x coordinates.

2. The result of step one is deducted from the real distance (For example, for x-coordinate - 1.20m)

Repeat the procedure for all modes and coordinates. It is noticeable that figure 2 contains only information on how accurate was the change of specific coordinates. For example, if Crazyflie was moved along the x-axis using TDoA3 mode, it can be seen that there was only a 1cm difference between estimated x-coordinates and the real distance.

| | x | y | z |
|---|---|---|---|
| TWR | **-0.16** | **0.08** | **0.03** |
| TDoA2 | **0.06** | **-0.07** | **0.02** |
| TDoA3 | **0.01** | **-0.06** | **0.03** |

Figure 2: results of distance accuracy measurements

Figure 3 shows how much difference there was between the coordinates. Bolded coordinates represent the axis on which Crazyflie moved. Non-bolded numbers are subtraction of two estimated coordinates, which should be 0 because Crazyflie was moved only along the bolded axis.

| | **x** | y | z |
|---|---|---|---|
| TWR | **-0.16** | 0.01 | -0.10 |
| TDoA2 | **0.06** | -0.02 | -0.16 |
| TDoA3 | **0.01** | 0.00 | -0.22 |
| | | | |
| | x | **y** | z |
| TWR | 0.12 | **0.08** | 0.00 |
| TDoA2 | 0.16 | **-0.07** | 0.11 |
| TDoA3 | 0.14 | **-0.06** | 0.14 |
| | | | |
| | x | y | **z** |
| TWR | 0.19 | 0.03 | **0.03** |
| TDoA2 | 0.19 | -0.02 | **0.02** |
| TDoA3 | 0.18 | 0.11 | **0.03** |

Figure 3: All differences between estimated distances and real distances.

### 4.1.4. Interpretation of results

Figure 1 shows that values are mainly in the range of 10cm accuracy, which was provided. TWR method's z coordinate is quite interesting because it is over 40 centimeters too big. TDoA2 and TDoA3 y- and z-coordinates are very accurate, but there is some error with x-coordinates.

Figure 2 shows that Crazyflie is able to measure 1D movement between two coordinates quite well. Only TWR x-coordinate had over 10cm error. Other coordinates errors were under 8cm.

Figure 3 shows other coordinates also, and it is pretty interesting that other coordinates are more inaccurate. Especially when moving along the y- and z-axis, all modes have over 10cm errors with x-coordinate. But since under 10cm accuracy was provided for a fixed position, the accuracy between

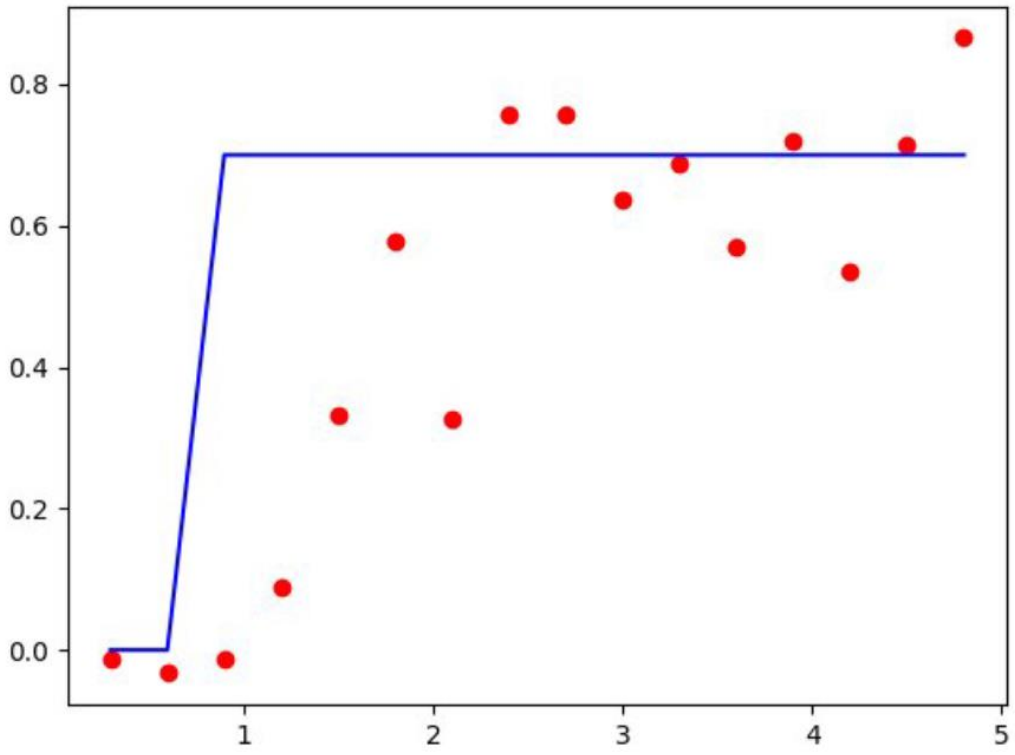
two coordinates should be under 20cm because the distance is a deduction of two fixed positions. So almost every value in figure 3 is in the promised range.

### 4.2. Velocity step response measurements

This test goal was to find out how quickly CF achieves given velocity. The test was arranged in Nebolab, and the test code is available in TODO. The test code included the following steps:

1. CF took off
2. 0.7m/s velocity was sent to the CF for four seconds in one direction.
3. Real velocity was calculated every 0.3 seconds, and the value was plotted to the graph

Real velocity was calculated as follows: $\frac{y_{t+\Delta t}-y_t}{\Delta t}$, where $\Delta t$ is 0.3sec and $y_t$ is the CF's estimated y-coordinate. X-axis values are in seconds, and y-axis values are m/s.




Picture 8: Velocity's step response.

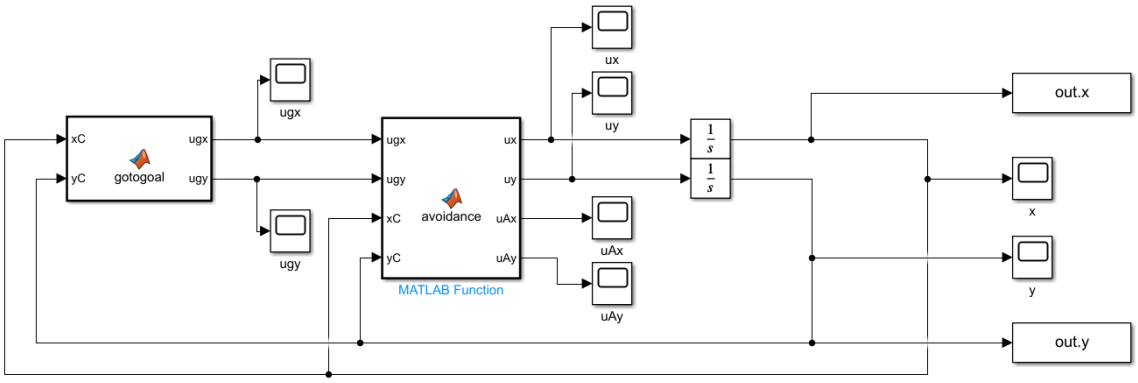
From picture 8, we can see that CF's rise time is approximately 1 second. There seems to be a little bit of overshooting and floating after the velocity is achieved, but because LPS is so inaccurate, I think that it is not interesting to calculate more detailed numbers.

I tried multiple tests with different sampling times ($\Delta t$), and 0.3 gave the best looking "curve". If sampling time was chosen too small, the range of values expanded significantly.

## 5. Simulations

### 5.1. Potential field path planning

I create a Simulink model of the system to simulate how go-to-goal and avoidance algorithms work and what could be suitable gains. The structure of the Simulink model is shown in picture 9.




Picture 9: Simulink model of the potential field path planning.

I use the following notation during section 5:

$V_g$ is go-to-goal potential function,
$V_{gx}$ is x-coordinate's go-to-goal potential function
$V_{gy}$ is y-coordinate's go-to-goal potential function
$x_c$ is current x-coordinate
$x_g$ is goal x-coordinate
$y_c$ is current y-coordinate
$y_g$ is goal y-coordinate

$U_g$ is go-to-goal control
$U_{gx}$ is x-coordinate's go-to-goal control
$U_{gy}$ is y-coordinate's go-to-goal control

$V_a$ is avoidance potential function
$V_{ax}$ is x-coordinates avoidance potential functions
$V_{ay}$ is y-coordinates avoidance potential functions
$q_c$ is current position of CF (x- and y-coordinates)
$q_o$ is current position of obstacle (x- and y-coordinates)
$R$ is avoidance function activate distance. When distance between obstacle and agent is smaller than $R$ avoidance function is activated.
$r$ is minimum safe distance. Obstacles must not be closer than this

$U_a$ is avoidance control
$U_{ax}$ is x-coordinate's avoidance control
$U_{ay}$ is y-coordinate's avoidance control

Go-to-goal potential function :

$$V_g = \begin{bmatrix} V_{gx} \\ V_{gy} \end{bmatrix} = \begin{bmatrix} \|x_c - x_g\|^2 \\ \|y_c - y_g\|^2 \end{bmatrix} = \begin{bmatrix} (x_c - x_g)^2 \\ (y_c - y_g)^2 \end{bmatrix} \quad (1)$$

The negative gradient of potential function:

$$U_g = \begin{bmatrix} U_{gx} \\ U_{gy} \end{bmatrix} = \begin{bmatrix} \frac{-\partial V_{gx}}{x_c} \\ \frac{-\partial V_{gy}}{y_c} \end{bmatrix} = \begin{bmatrix} -(2 * x_c - x_g) \\ -(2 * y_c - y_g) \end{bmatrix} = \begin{bmatrix} -2(x_c - x_g) \\ -2(y_c - y_g) \end{bmatrix} = \begin{bmatrix} 2x_g - 2x_c \\ 2y_g - 2y_c \end{bmatrix} \quad (2)$$

Obstacle avoidance potential function:

$$V_a = \left(\min\left\{0, \frac{\|q_c - q_o\|^2 - R^2}{\|q_c - q_o\|^2 - r^2}\right\}\right)^2 \quad (3)$$

The negative gradient of potential function:

$$U_a = \begin{bmatrix} U_{ax} \\ U_{ay} \end{bmatrix} = \begin{bmatrix} \frac{-\partial V_{ax}}{q_c} \\ \frac{-\partial V_{ay}}{q_c} \end{bmatrix} = \begin{bmatrix} \frac{4(R-r)^2(\|q_c - q_o\|^2 - R^2)(x_c - x_o)}{(\|q_c - q_o\|^2 - r^2)^3} \\ \frac{4(R-r)^2(\|q_c - q_o\|^2 - R^2)(x_c - x_o)}{(\|q_c - q_o\|^2 - r^2)^3} \end{bmatrix}, \text{ when } r < \|q_c - q_o\| < R \quad (4)$$

If the condition of formula 4 is not met, then $U_a = 0$.

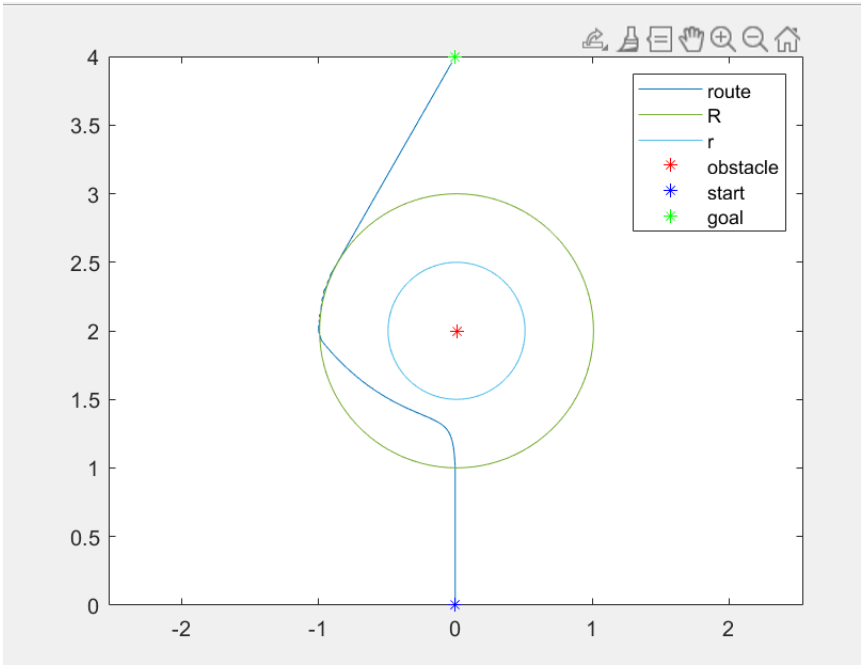
Final control velocity is the sum of formulas (2) and (4) multiplied with P-gains:

$U = K_g U_g + K_a U_a$ , where $K_a$ is presented in formula 6. (5)

$$K_a = \frac{constant}{\sqrt{|U_{ax}^2 + U_{ay}^2|}} \quad (6)$$

I found out that the suitable gain for the go-to-goal algorithm is 0.05 and scale constant to the avoidance algorithm 0.25.

Picture 10 shows one example case of how the algorithm work. The starting point of the CF is 0, 0, and obstacle coordinates are 0.01, 2. The goal is in 0, 4. Obstacle's and CF's x-coordinates are a little bit different to avoid the local minima problem.

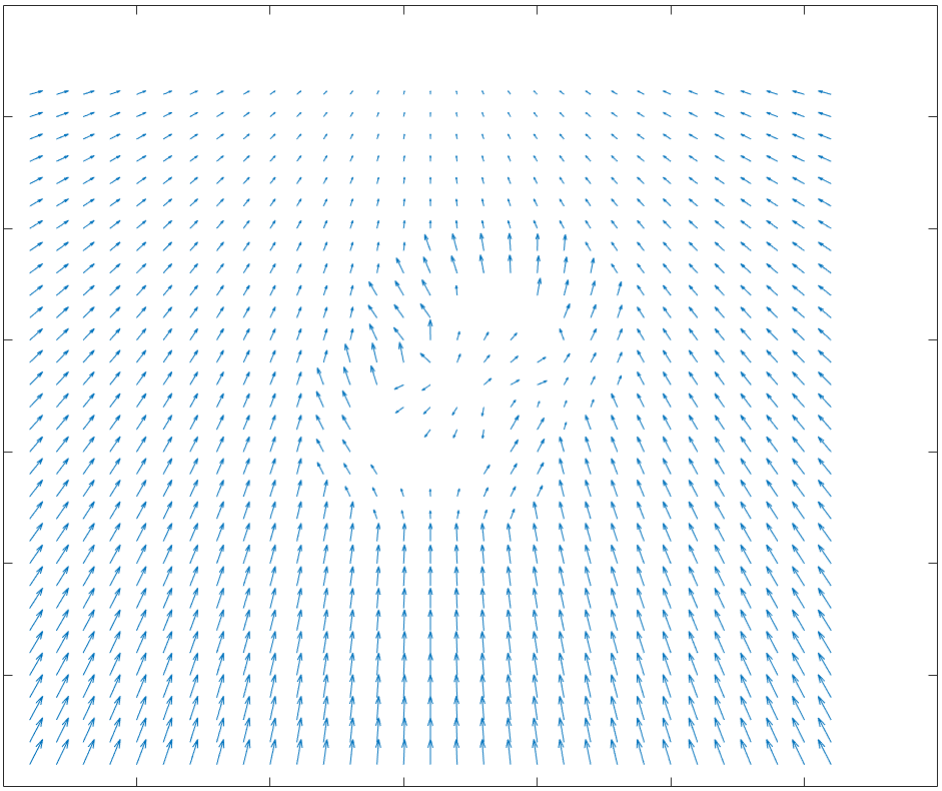



Picture 10: Test case simulation result.

We can see from the picture that when CF distance is less than R the avoidance function is activated, and CF starts to avoid the obstacle. In this test R=1, r=0.5 and sampling time is 0.1s.

## 5.2. Vector field

Vector field shows how potential changes in the area under consideration.



Picture 11: Vector field presentation, two obstacles.

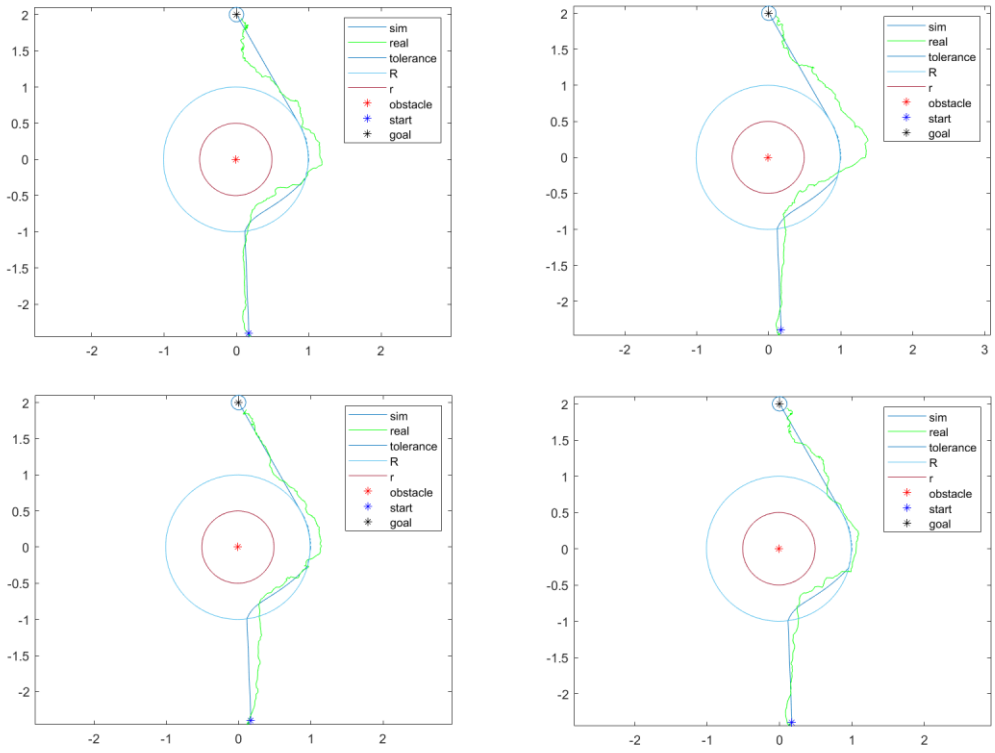
Picture 11 shows one example of a vector field. At the bottom of the picture, it can be seen that only the go-to-goal algorithm is activated. When the agent is approaching obstacles, the avoidance algorithm begins to affect more and more. The direction of the vectors turns away from the obstacles, and the agent tries to bypass the obstacle. At the top of the picture, only low go-to-goal force is affected.

## 5.3. Tests

This section includes different test cases, where simulation and real route are presented in the same picture.

### 5.3.1. One static obstacle

I run multiple tests with CF in Nebolab. Picture 12 shows four tests, where starting point, obstacle and goal point were the same.

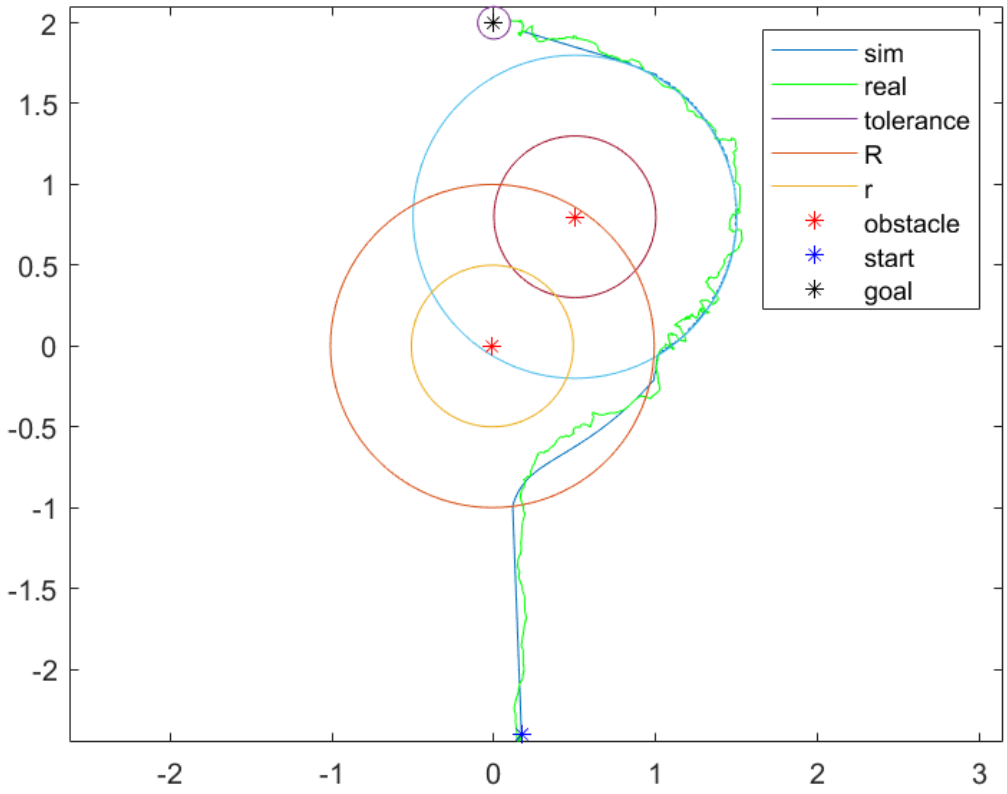



Picture 12: Test results of path planning with one static obstacle.

As can be seen, CF managed to avoid the obstacle and achieve the goal point in all cases. There was some variance in the routes, but in no case did the CF get inside the forbidden area.

### 5.3.2. Multiple static obstacles

In this test case, multiple static obstacles can be added to the simulation. Picture 13 shows a case where there are two static obstacles, but they are so close to each other's that there is no deadlock.




Picture 13: One test result with multiple static obstacles.

In this case, the real route and simulation are very close to each other's, and the goal is achieved.