UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Intelligent Robotics

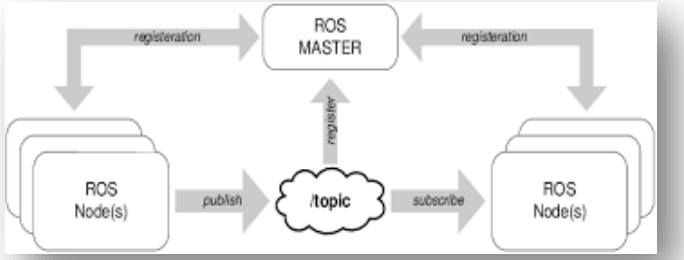
Group 23 - Assignments Presentation

Simone Bastasin, Alessandro Manfè, Andrew Zamai

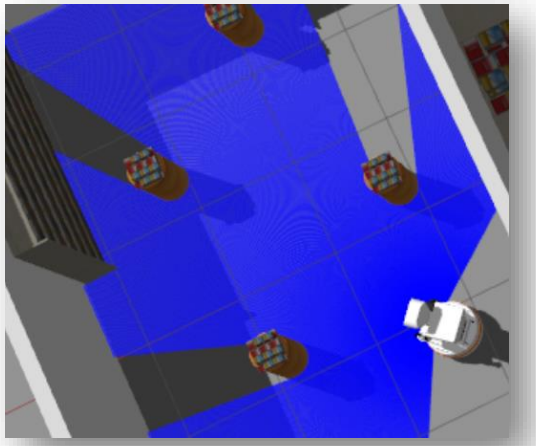
Topics we will focus on

Assignment 1

- ROS nodes architecture design

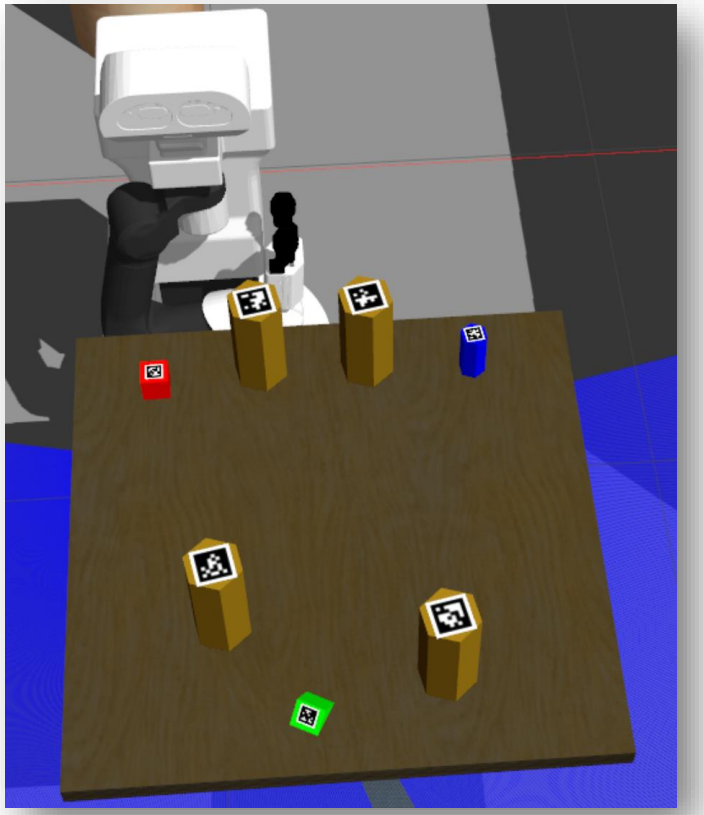


- Algorithm for detecting mobile obstacle centers



Assignment 2

- Pick and Place routine



ROS nodes architecture design

- Guidelines to follow: write **structured**, **modular** and **easily extendable** code
- **Exploit** ROS **nodes architecture**
- Implement different programs in different nodes
- make them **communicate** through **messages**, **services** and **client/server actions**

HOW DOES THIS TRANSLATE INTO ASSIGMENT 1?

translating it to assigment 1

- From assignment specifications:
  - the action client receives the input from the user;
  - the action client calls the **action server** that **executes all the tasks**;
  - the action server sends the final list of positions of the obstacles as result to the action client;
  - move_base node to be used for navigation

- Then **just 2 nodes are required**, additionally to the already existing ***move_base*** node:
  - Action client:

  receives in input goal Pose_B and publishes back periodically Tiago's current status + detected obstacle centers
  - Action server:

  dispatches goal pose to move_base node and perform obstacle centers detection

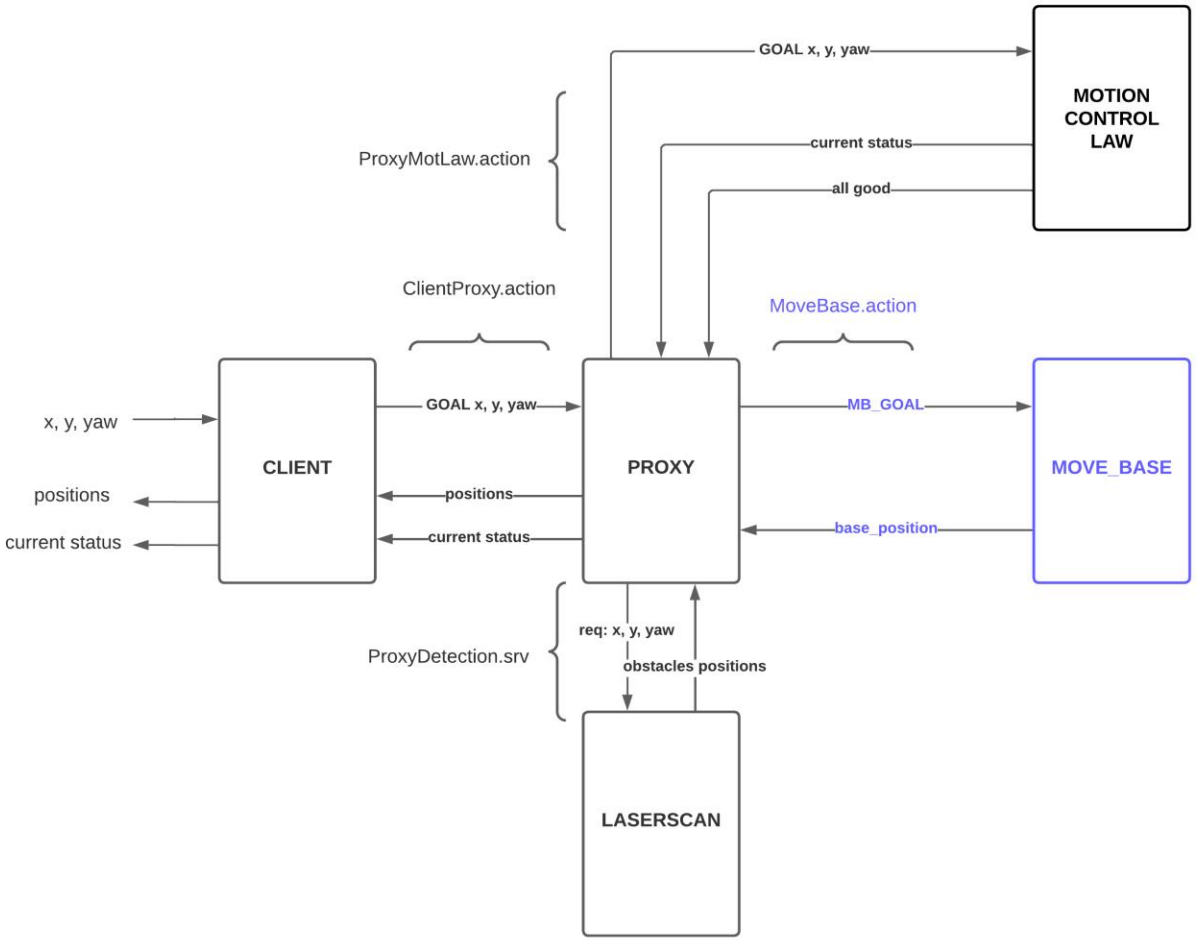
CAN WE DO BETTER IN TERMS OF MODULARITY AND FUTURE EXTENDABILITY?

improving modularity and extendability

- **Problems:**
  - Action server is required to handle **both** navigation and obstacles detection **tasks**
  - Extra points require to **add more code to action server** to implement a naive Motion Control Law through the corridor

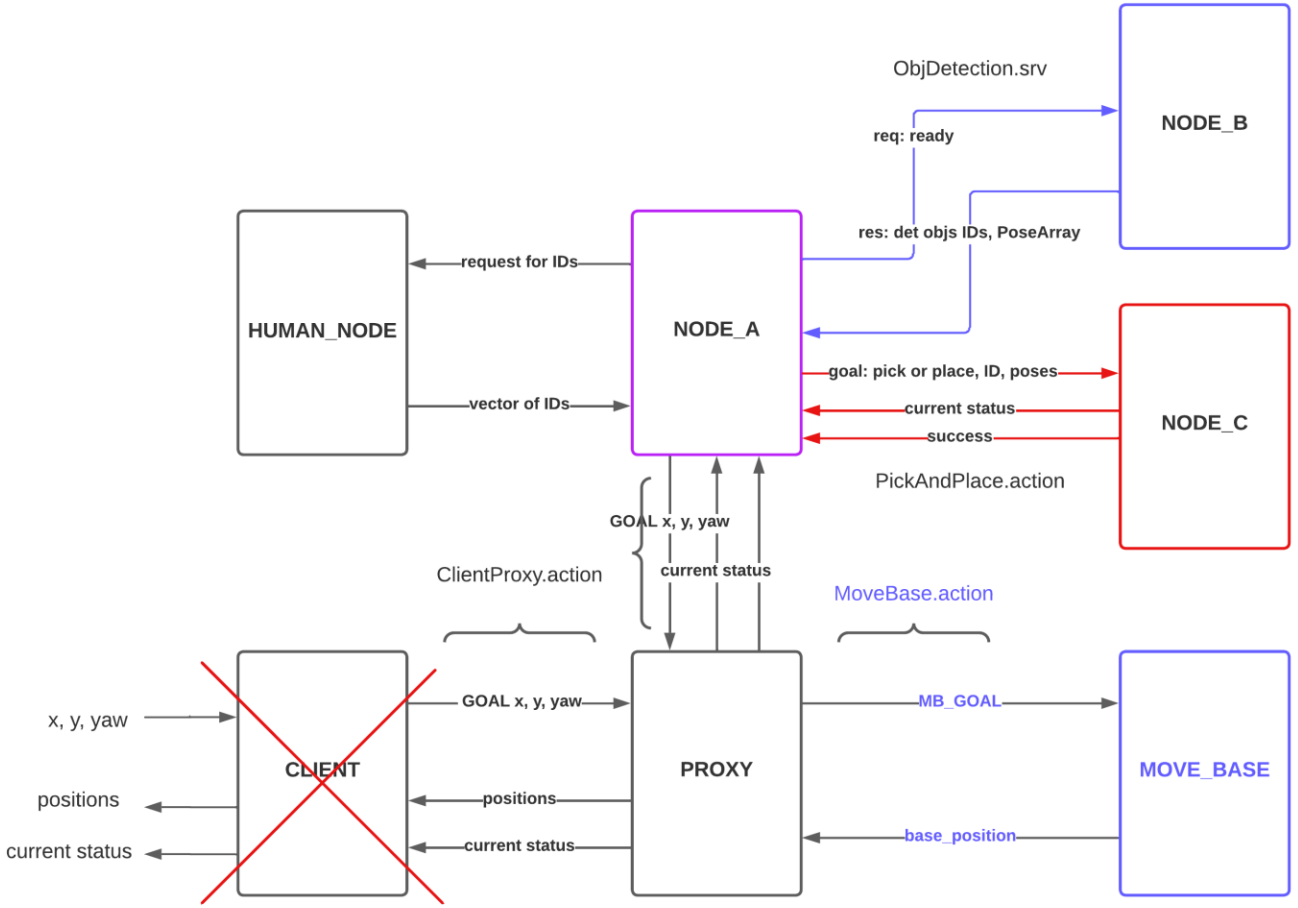
- **Solutions:**
  - **Split** navigation handling and obstacles detection task **in 2 different nodes**
  - Create an **additional node** that **implements the naive MCL**
  - Create a node that acts as **middleman** between the action client and all these nodes: the ***PROXY***

Final ROS nodes architecture design



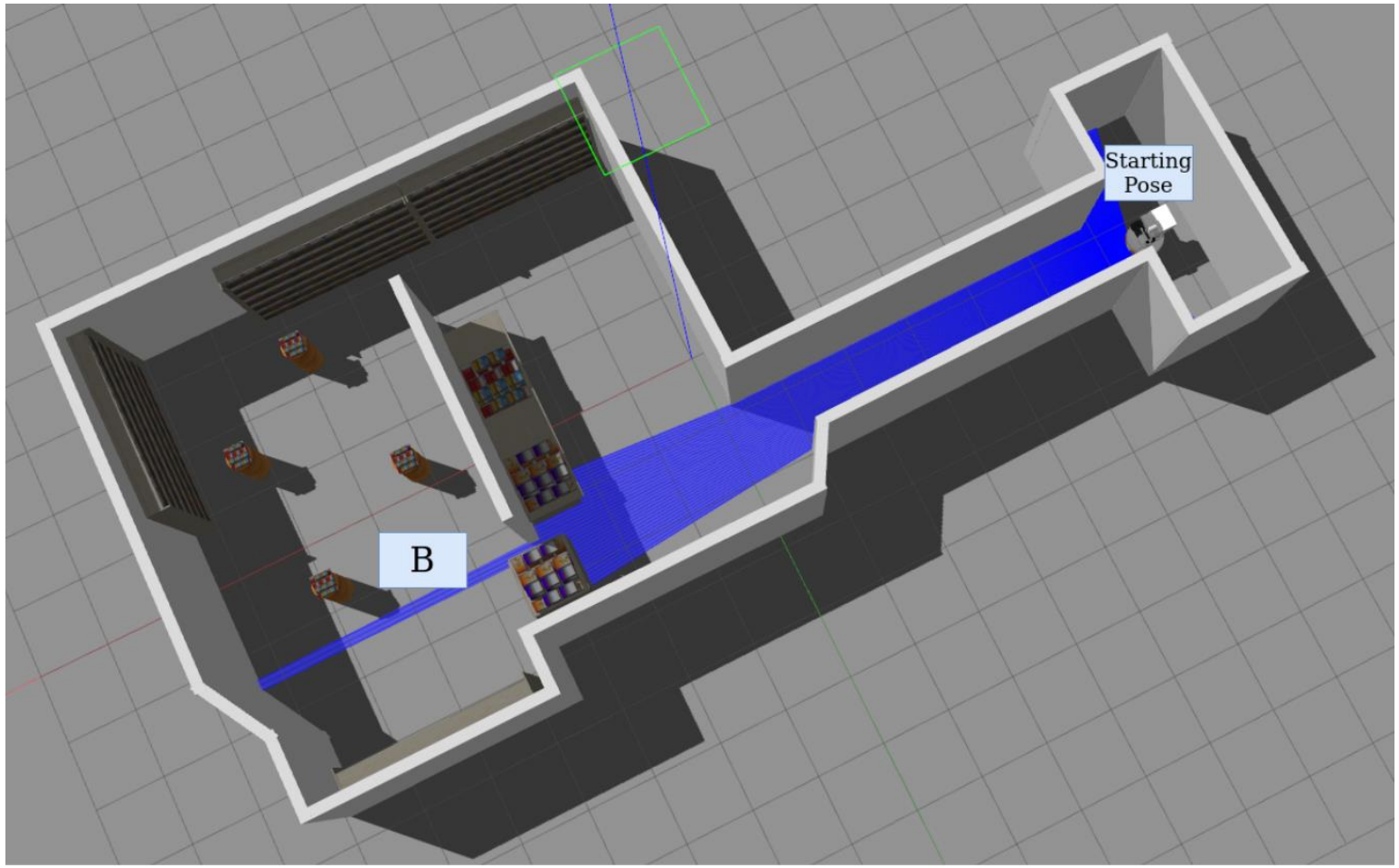
Middleman node acting as ***proxy*** gets the goal from the client and dispatch it firstly towards the motion control law server, then to the move base server and lastly sends the request for the detection to LaserScan node

Showing ease of extension in assignment 2



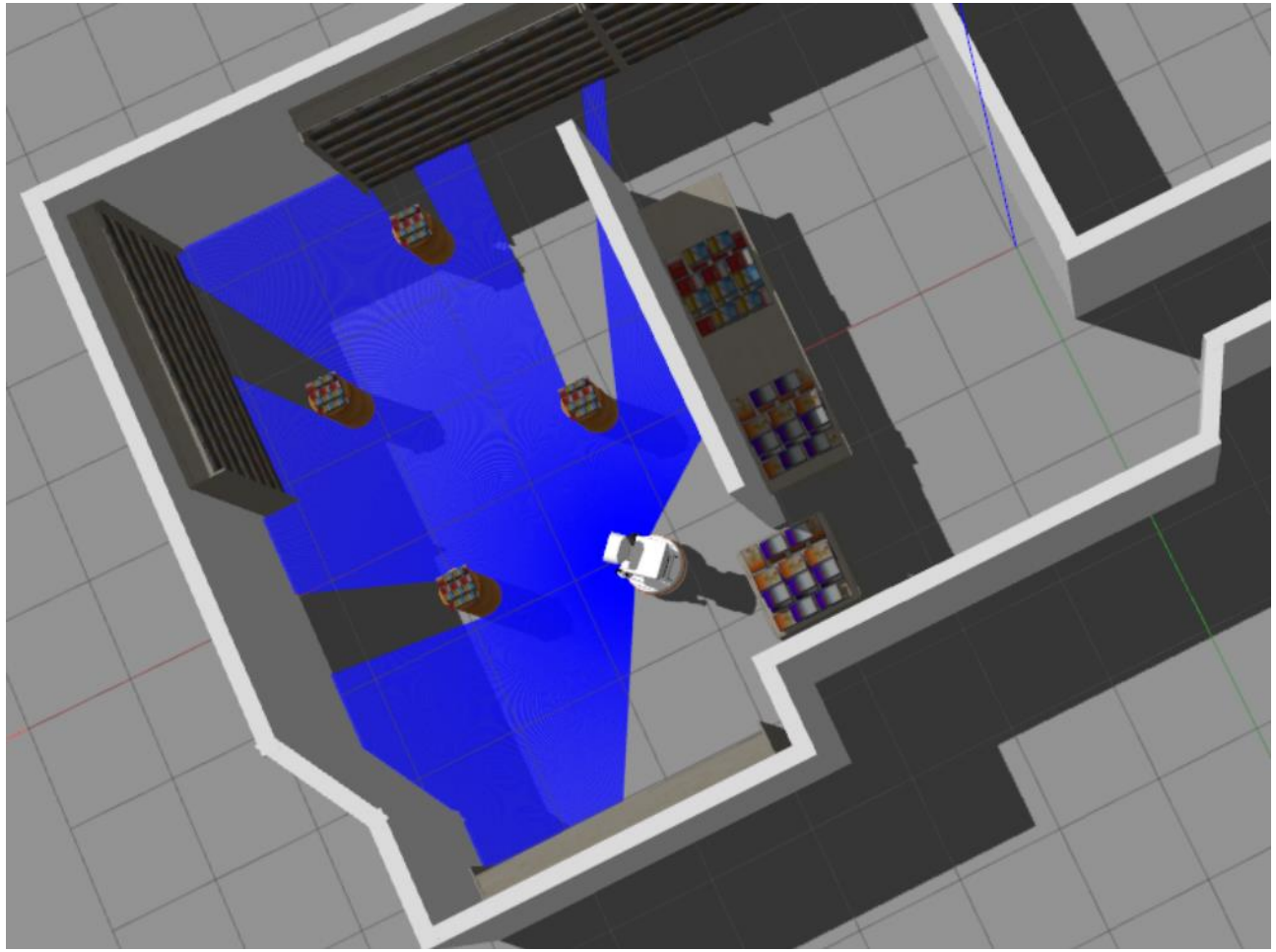
Assigment 1 navigation can be re-used in assignment 2, without having to change the previous code, by simply making the new nodeA communicate with the Proxy through ClientProxy.action, similarly as the Client node was doing

Obstacles detection



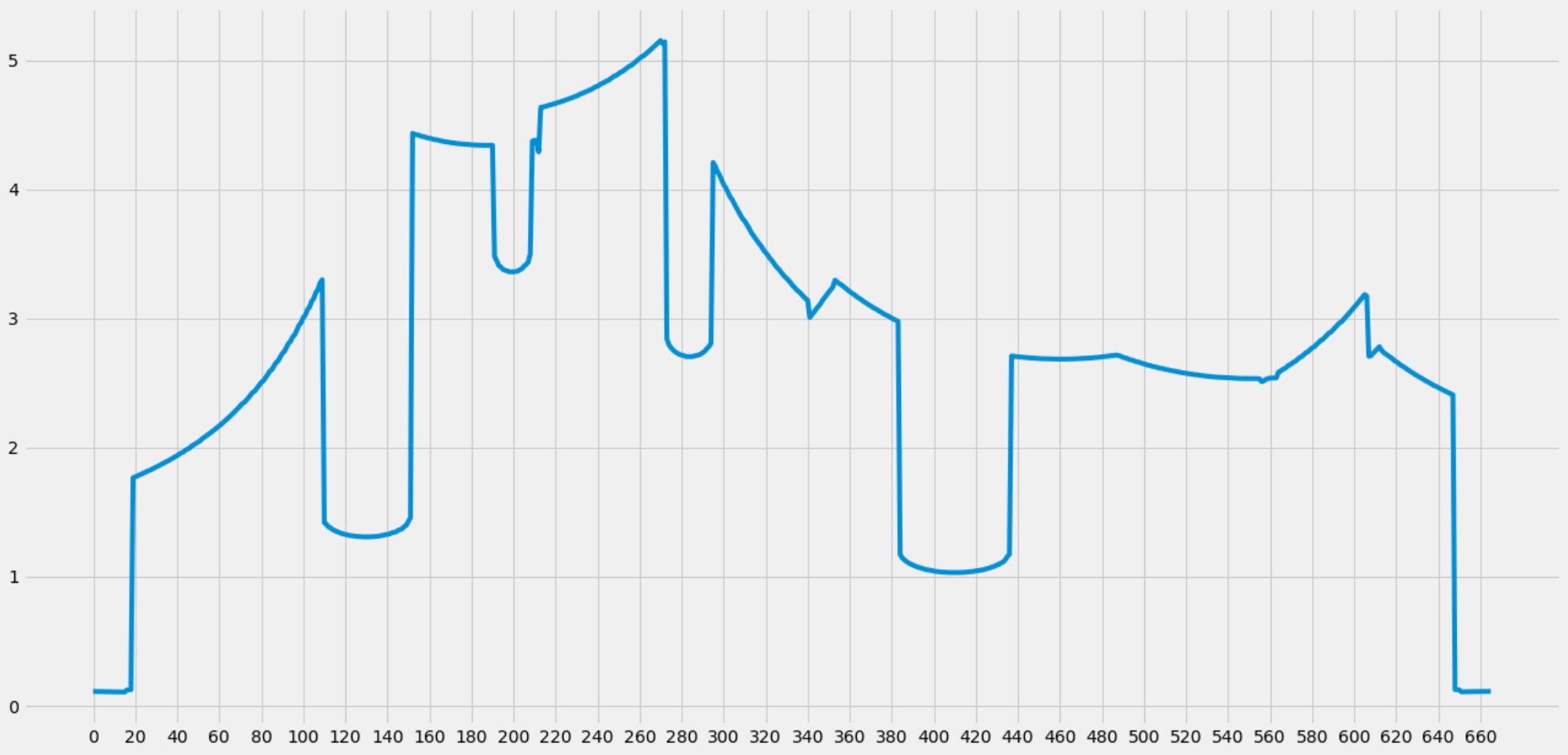
- **1st step**: move from *Starting Pose* to *Pose_B*

Obstacles detection



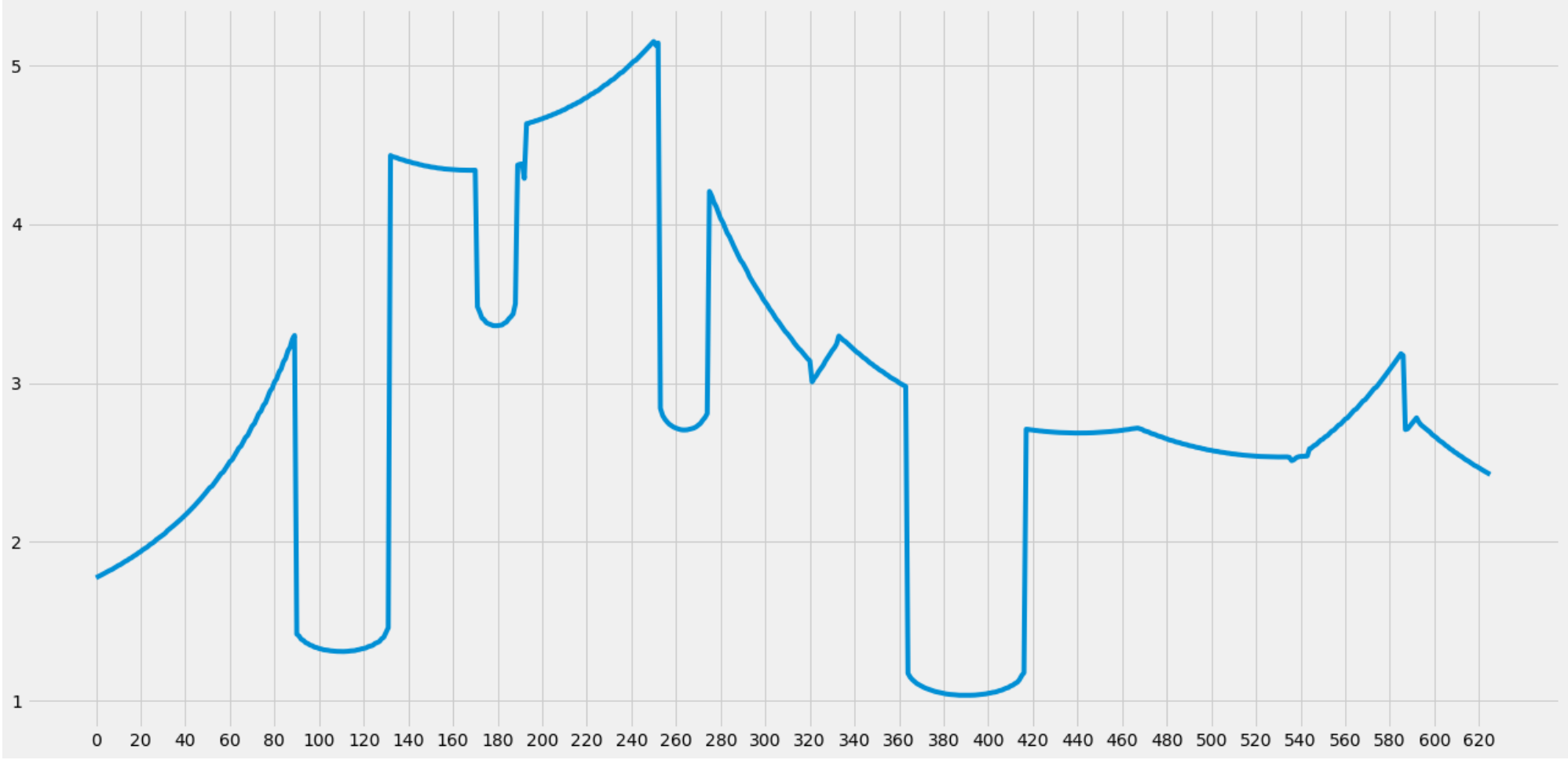
- **2nd step**: scan environment through "*/scan*" topic

Obstacles detection



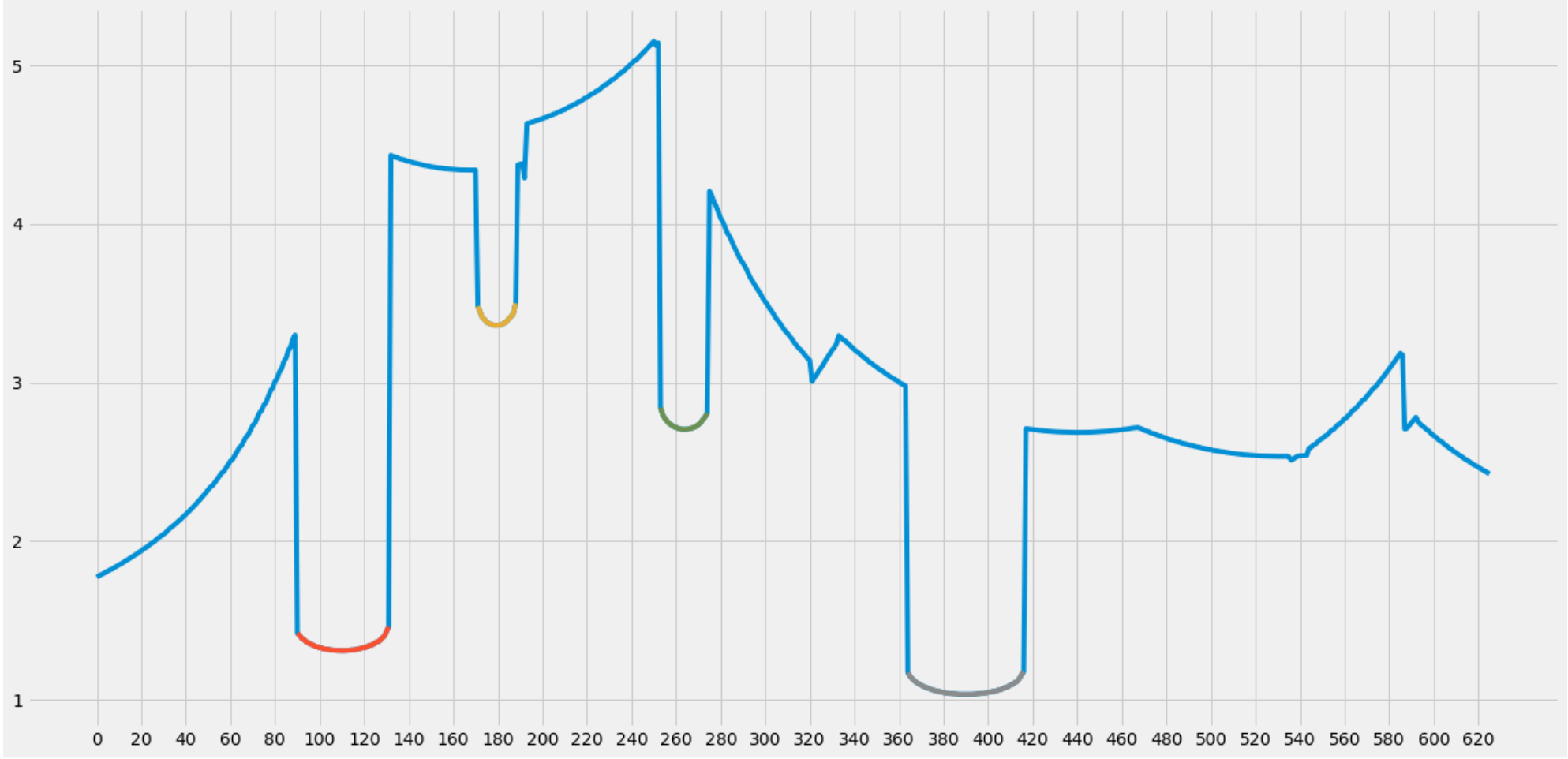
- **3rd step**: evaluate vector of ranges

Obstacles detection



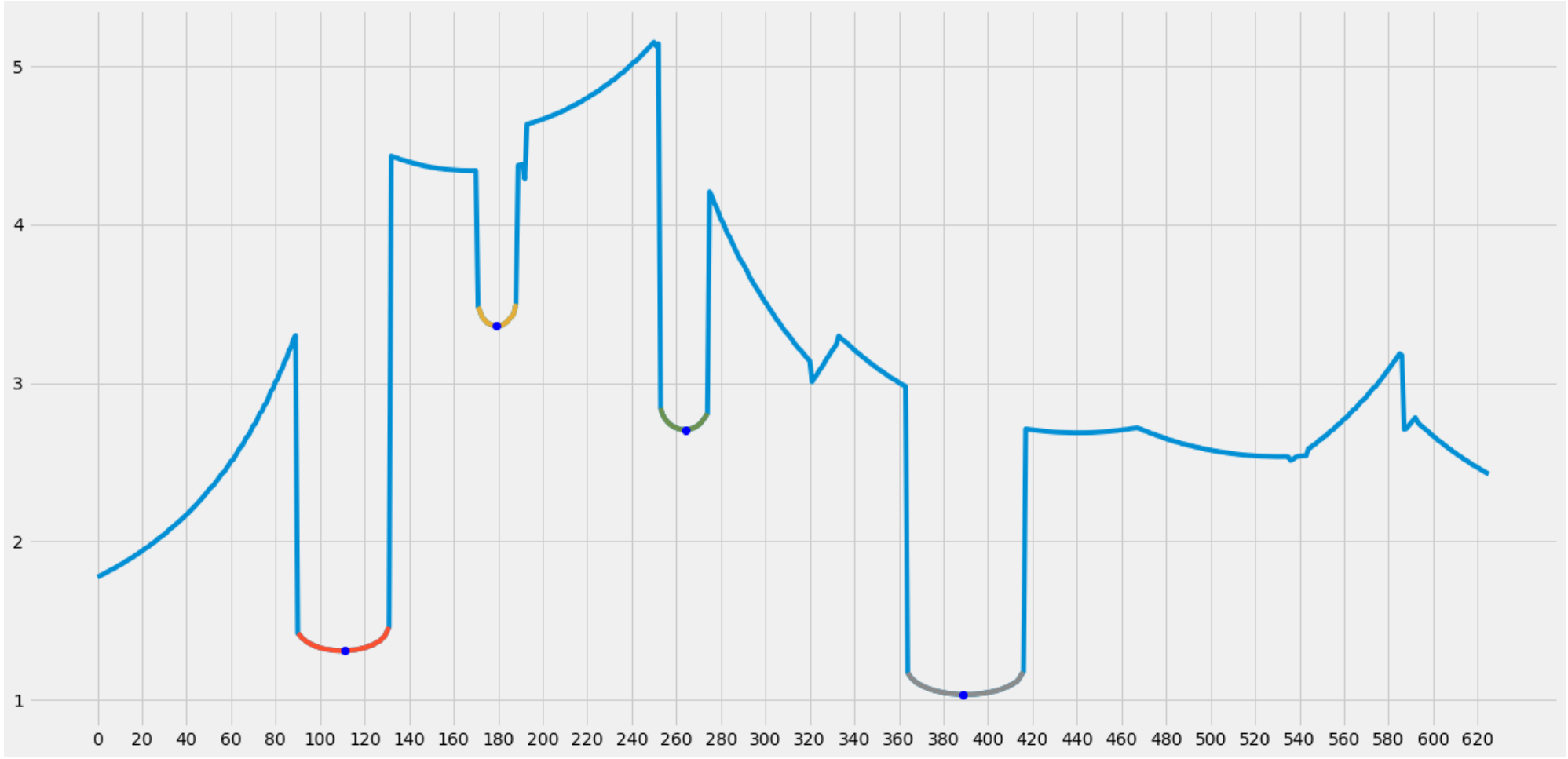
- **4th step**: remove noisy values

Obstacles detection



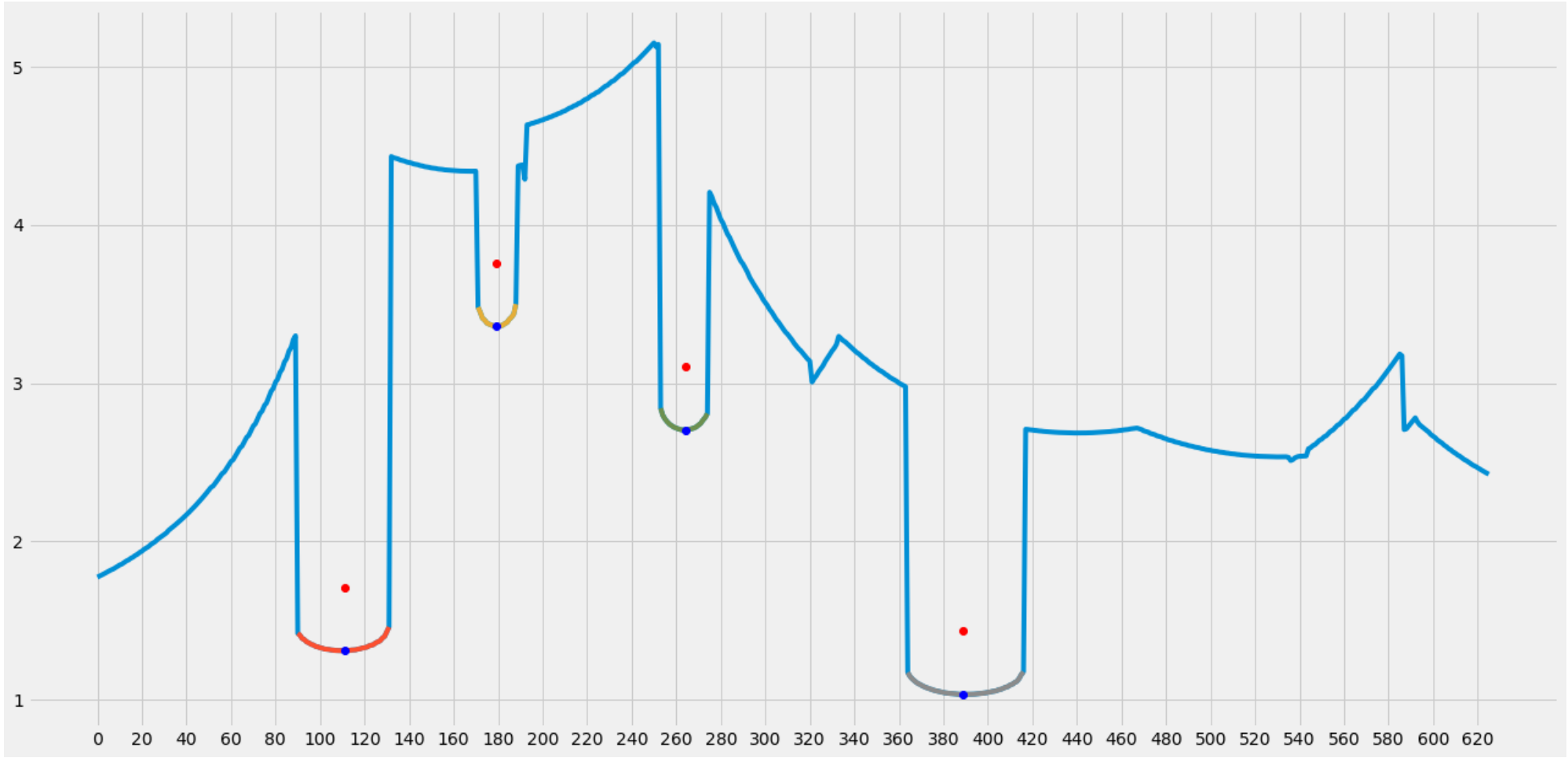
- **5th step**: find depressions (corresponding to obstacles)

Obstacles detection



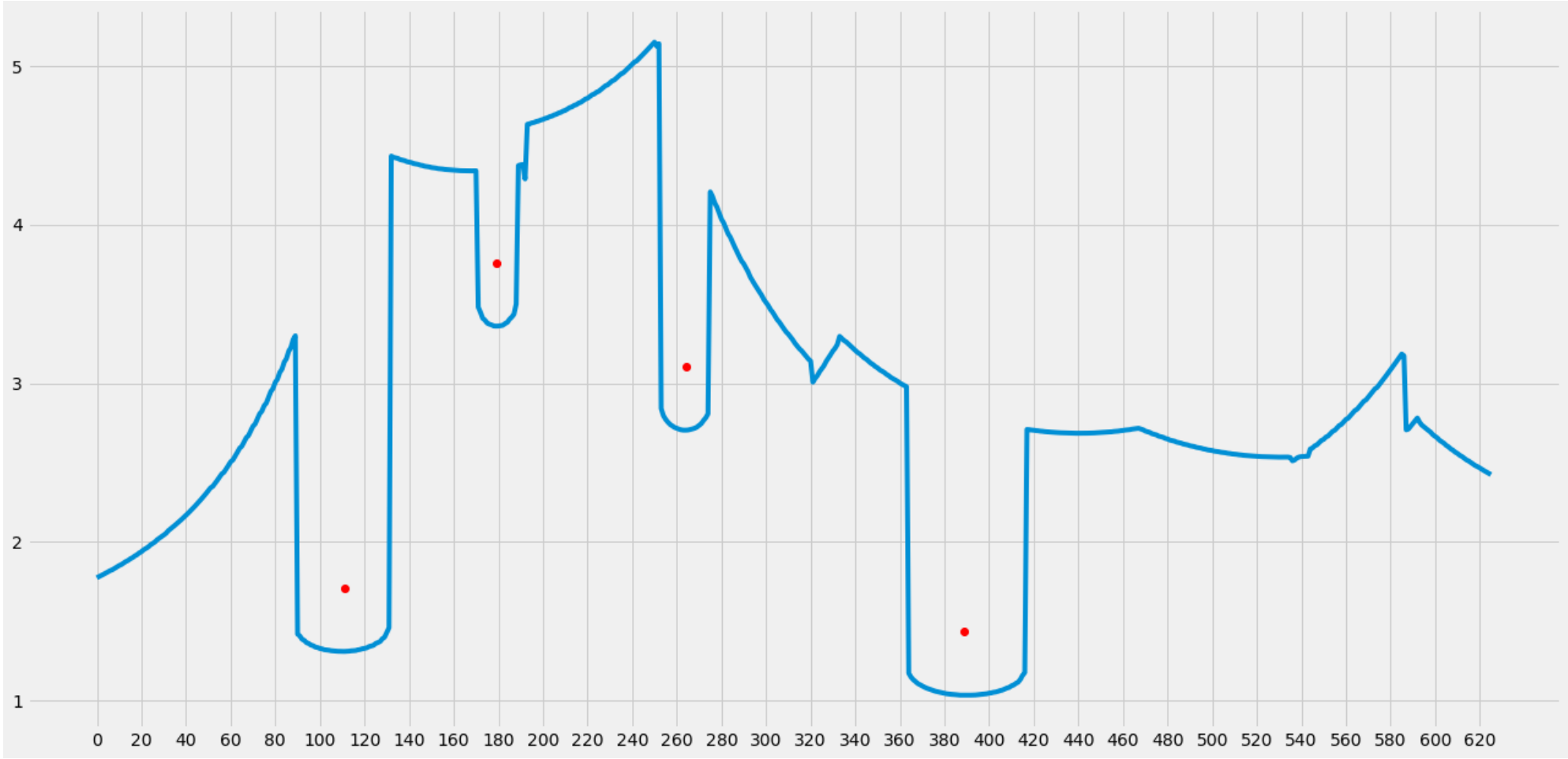
- **6th step**: find min in depressions (closest points of obstacles to robot)

Obstacles detection



- **7th step**: from edge point to center point

Obstacles detection



- **8th step**: final centers in polar coordinates

Obstacles detection

- **9th step**: from polar coordinates to cartesian coordinates:
  - $(r, \varphi) \rightarrow (x, y)$

- $x = \mathbf{cos}(angle\_min + \varphi * angle\_increment) * r$
- $y = \mathbf{sin}(angle\_min + \varphi * angle\_increment) * r$

THESE NEW COORDINATES ARE W.R.T. ROBOT REFERENCE FRAME

Obstacles detection

- **10th step**: from robot reference frame to world reference frame:
  - (x, y) w.r.t. robot r.f. → (x', y') w.r.t. world r.f.
- Transformation matrix computed manually (not using *tf library*)
  - *angle*, *dx* and *dy* depend on *Pose_B* w.r.t. to "*/map*" r.f.y

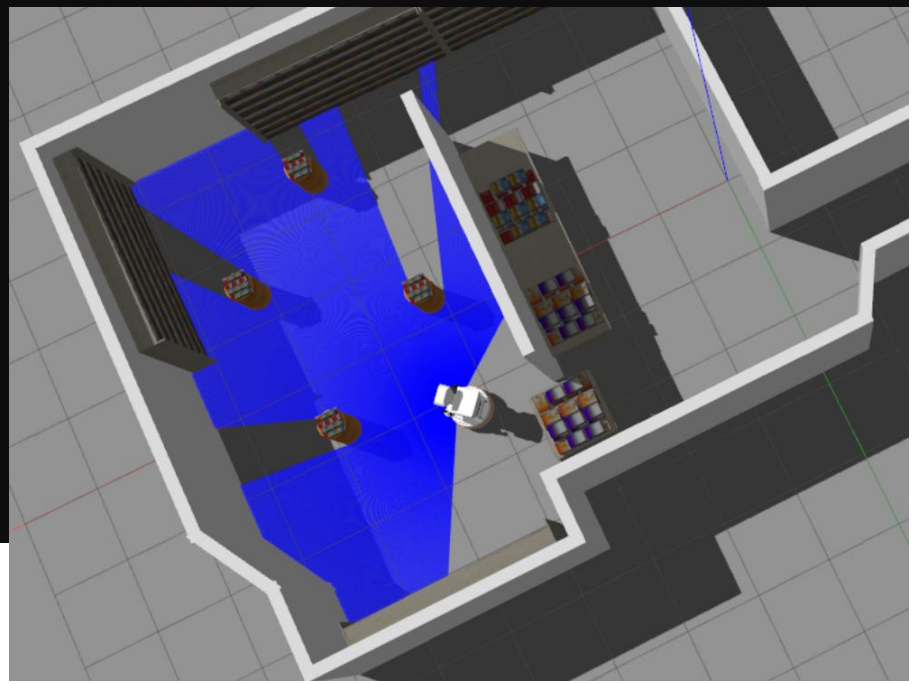
- $x' =$
- $y' =$

$$\begin{bmatrix} \cos(angle) & -\sin(angle) & 0 & dx \\ \sin(angle) & \cos(angle) & 0 & dy \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

NOW WE HAVE COORDINATES W.R.T. WORLD REFERENCE FRAME

Obstacles detection

```
[ INFO] [1671821504.895009200, 4099.224000000]: Tiago status: the robot is moving
[ INFO] [1671821505.026293900, 4099.323000000]: Tiago status: the robot is moving
[ INFO] [1671821505.028629900, 4099.324000000]: Tiago status: the robot started the detection of the obstacles
[ INFO] [1671821505.383702100, 4099.581000000]: Tiago status: the detection is finished
[ INFO] [1671821505.383798300, 4099.581000000]: Tiago successfully detected 4 obstacles from Pose_B
[ INFO] [1671821505.383827500, 4099.581000000]: Detected obstacles positions (x, y) w.r.t Robot reference frame:
[ INFO] [1671821505.383844900, 4099.581000000]: (0.764954, -1.616565)
[ INFO] [1671821505.383856200, 4099.581000000]: (2.815837, -2.641740)
[ INFO] [1671821505.383866300, 4099.581000000]: (3.098320, -0.871731)
[ INFO] [1671821505.383876100, 4099.581000000]: (1.397813, 0.641344)
[ INFO] [1671821505.383886900, 4099.581000000]: Detected movable
obstacles positions (x, y) w.r.t World reference frame:
[ INFO] [1671821505.383899800, 4099.581000000]: (3.817829, -0.314008)
[ INFO] [1671821505.383909300, 4099.581000000]: (4.543114, -2.489108)
[ INFO] [1671821505.383919300, 4099.581000000]: (5.994446, -1.437269)
[ INFO] [1671821505.383928900, 4099.581000000]: (5.861912, 0.835077)
simon@LAPTOP-E9C0TNE2:~/tiago_public_ws$
```



- **11th step**: final results
  - Real centers: (4.155, -0.295), (4.875, -2.424), (6.160, -1.347), (5.811, 0.763)

Pick and Place

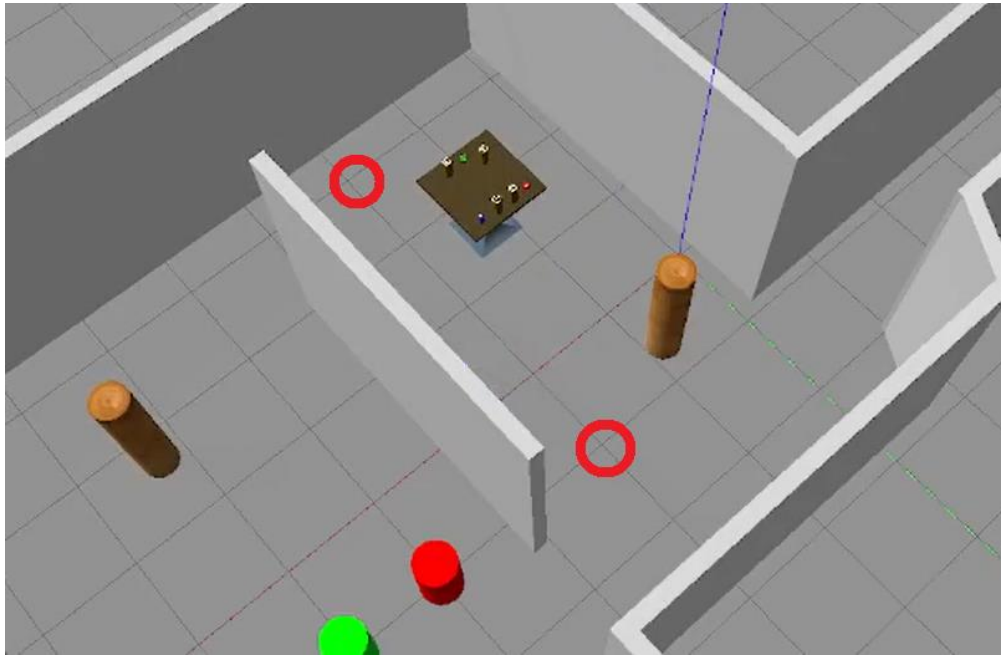
- Node C :
  - Action Server
  - Goal: pickOrPlace, poses from detection
  - Fulfills requests of pick and place differently
  - Combination of joint space and cartesian movements

Pick and Place

Problem: Unwanted collision

Solution:

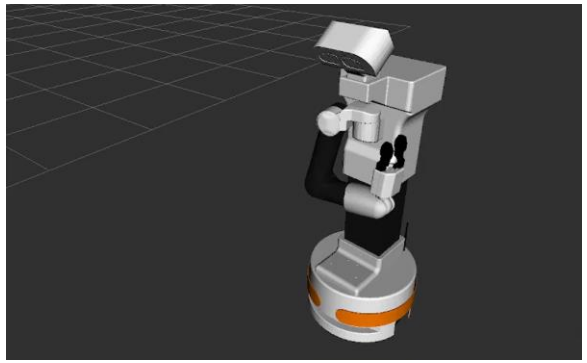
- Waypoints (during movement)
  - Better predictability of movements paths
- Safe position
- Bigger collision objects

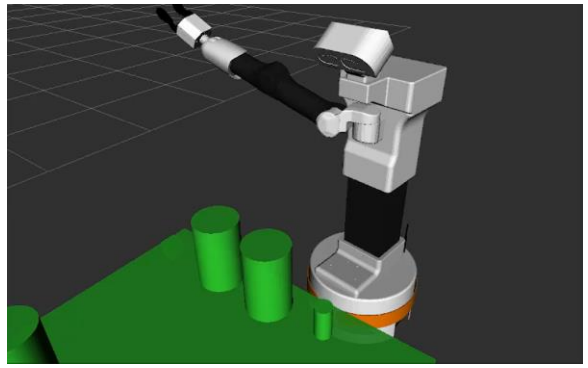


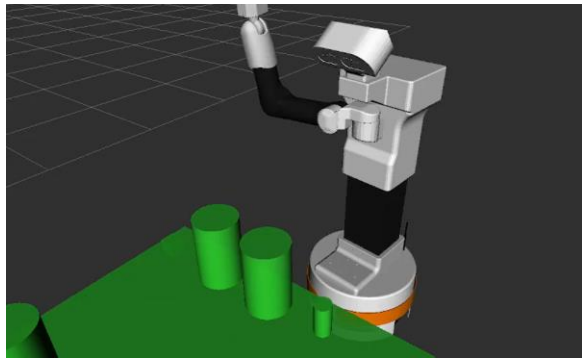
Pick and Place

Safe Position:

- Easier for the planner to compute a path
- Prevent unwanted collision due to misaligment





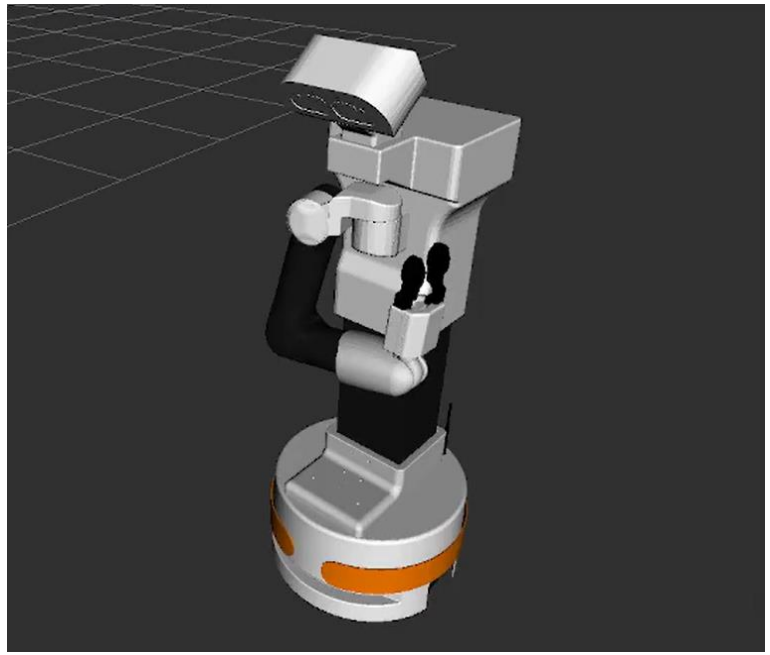


Pick and Place

Problem: Occasional detection errors (April Tag)

Solution: Torso lift and Head Tilt

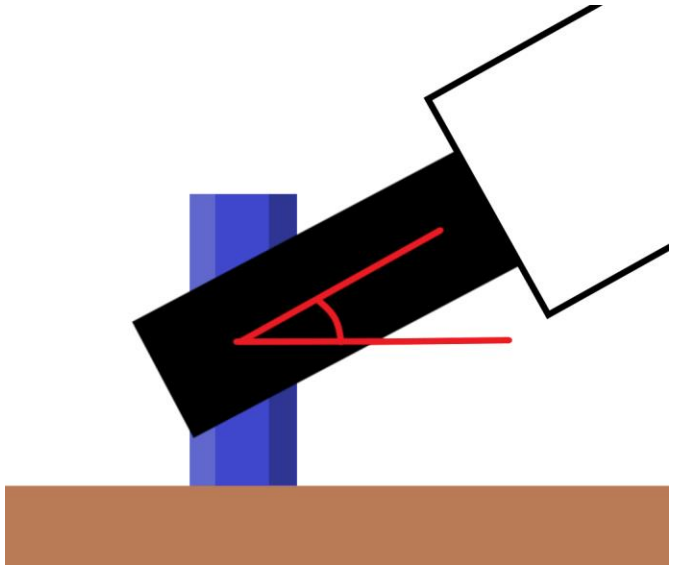
- Better camera position, shot and tag recognition

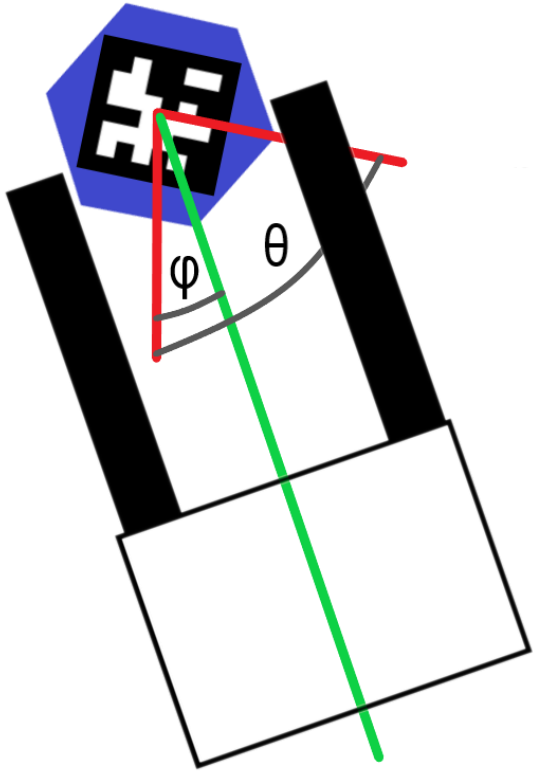


Pick and Place

Pick Action: random pose and best approach

- Yaw module
- Gripper pitch
- Pose correction for green triangle

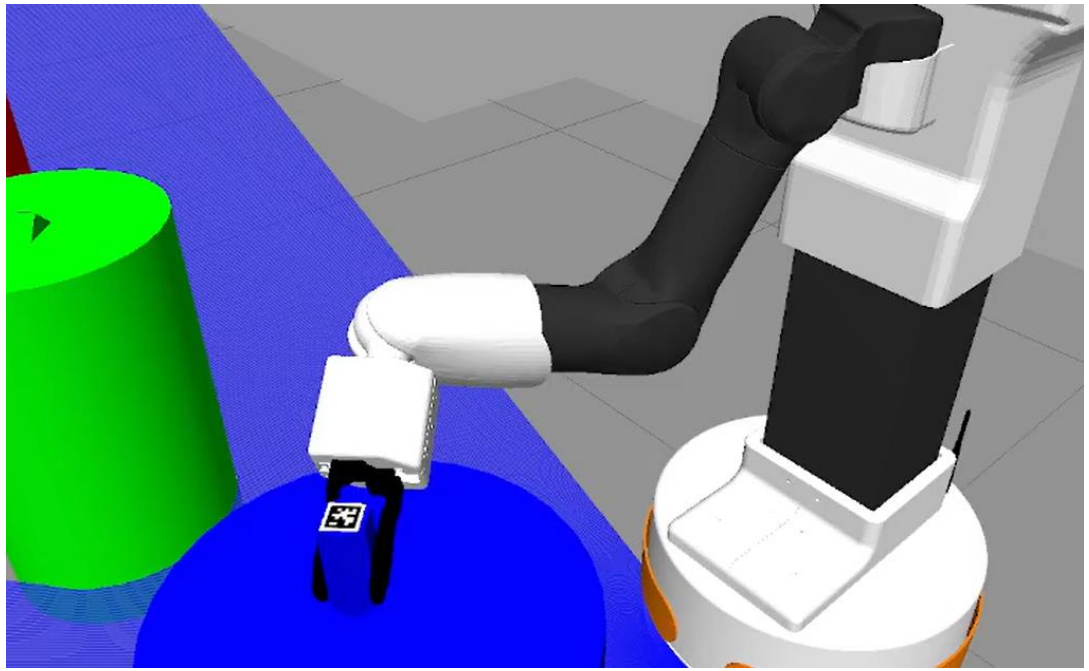




Pick and Place

Place Action:

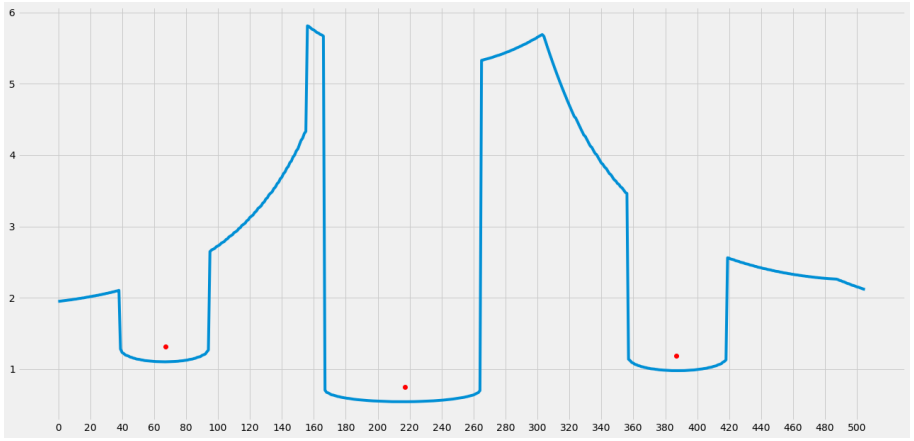
- Same setup as Pick (Safe position)
- Z-approach with few mm of clearance

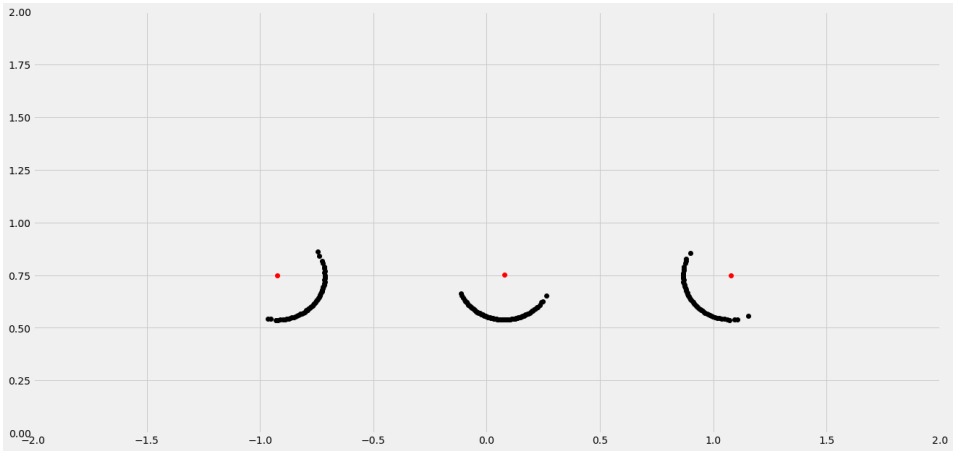


Pick and Place

Bonus Point: Laser scan cylindrical tables

- New waypoint for laserscan
- Same approach as assigment 1 (discarding extremes and finding depressions)
- 3-points centers Interpolation





Thanks for your attention