

---

---

# EE405A

# Final Project Preparation

(TA) Hyunki Seong, Gyuree Kang, Sungjae Min, Hyunjoo Kim  
School of Electrical Engineering  
KAIST

November. 24<sup>th</sup>, 2023

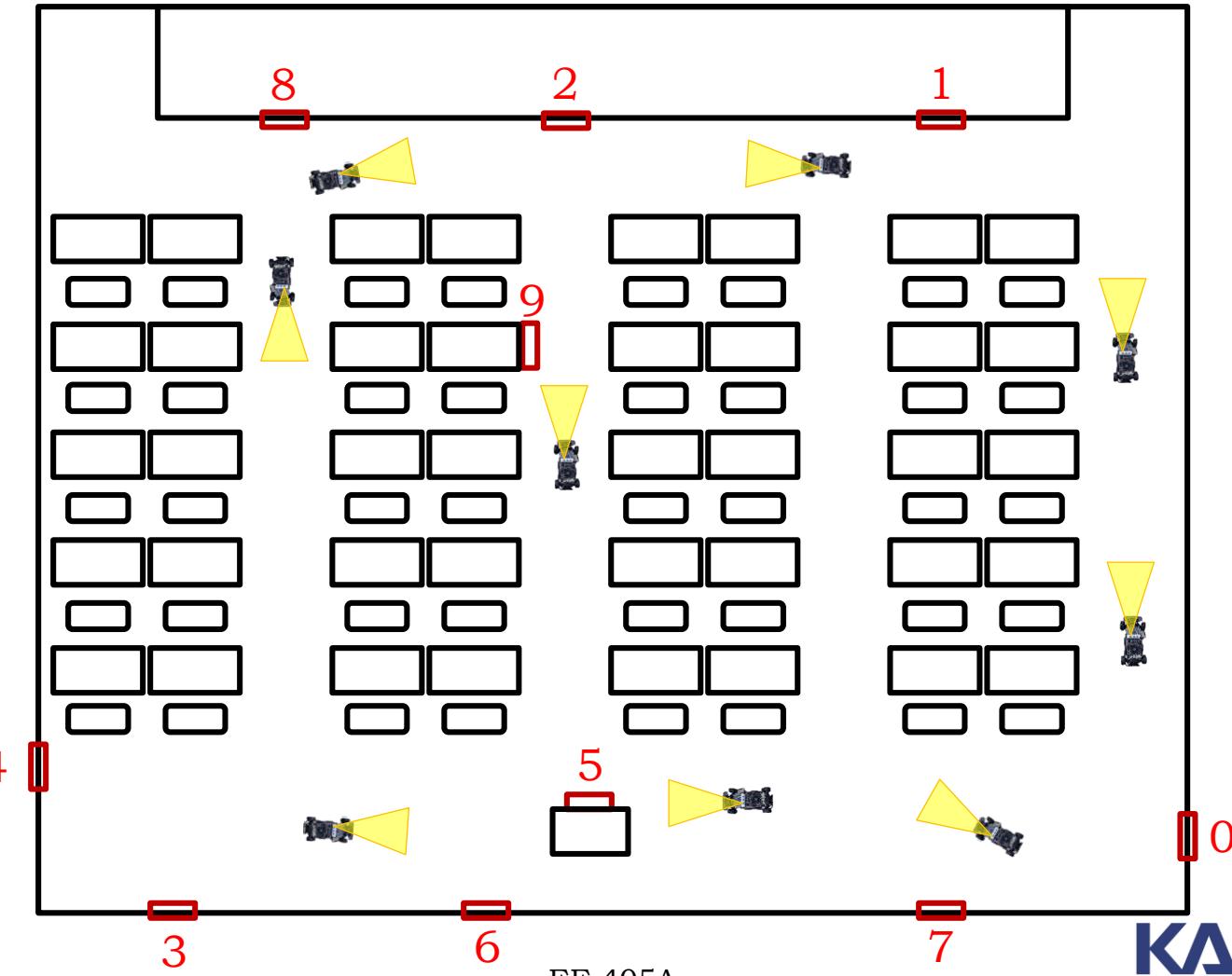
---

---

# Final Project Scenario

# Final Project Scenario

- ❖ Exploring an indoor environment to find target objects
  - **Target objects:** papers numbered 0 through 9 (example image).

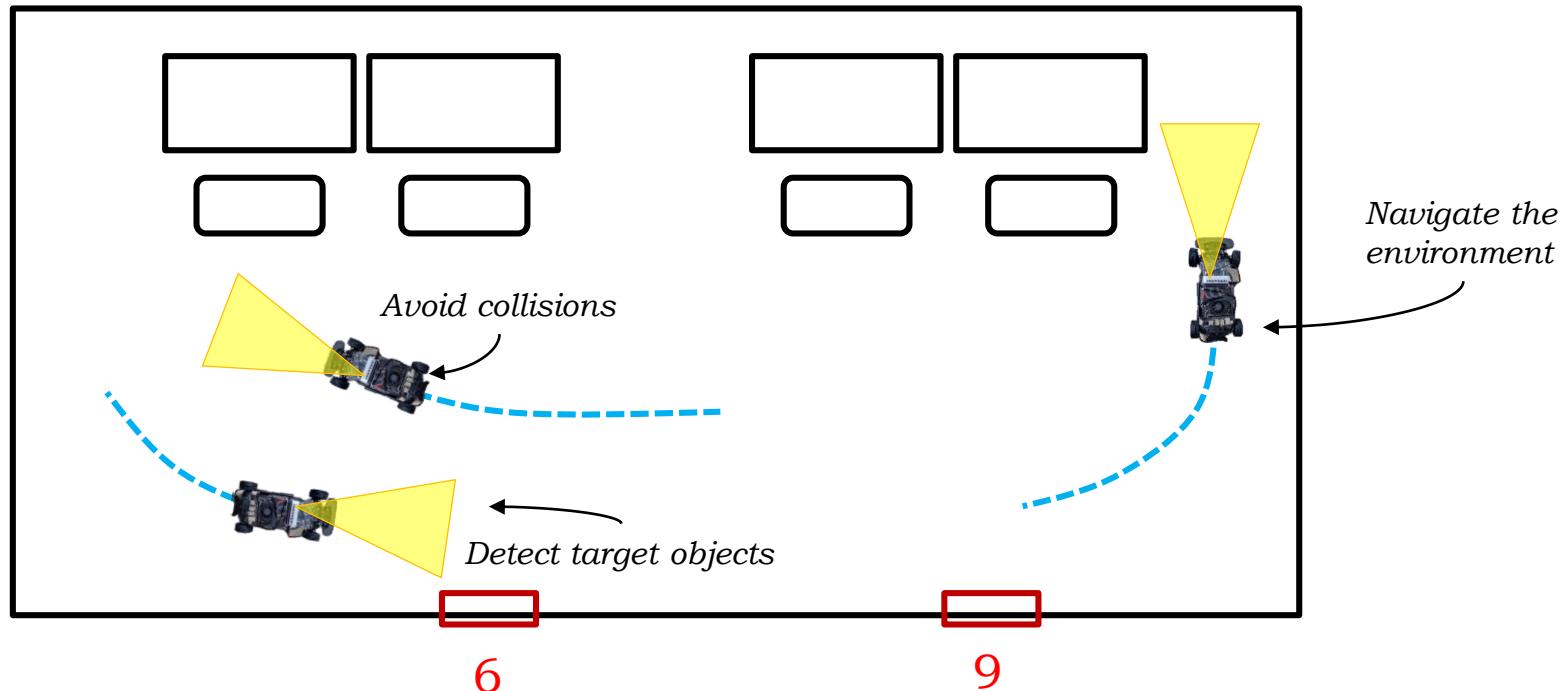


# Final Project Scenario

- ❖ Exploring an indoor environment to find target objects

- Main Tasks:

- Avoid collisions with other vehicles and walls (or desks)
    - Navigate the environment & Detect and Classify target objects
    - Estimate the global position of the detected target object



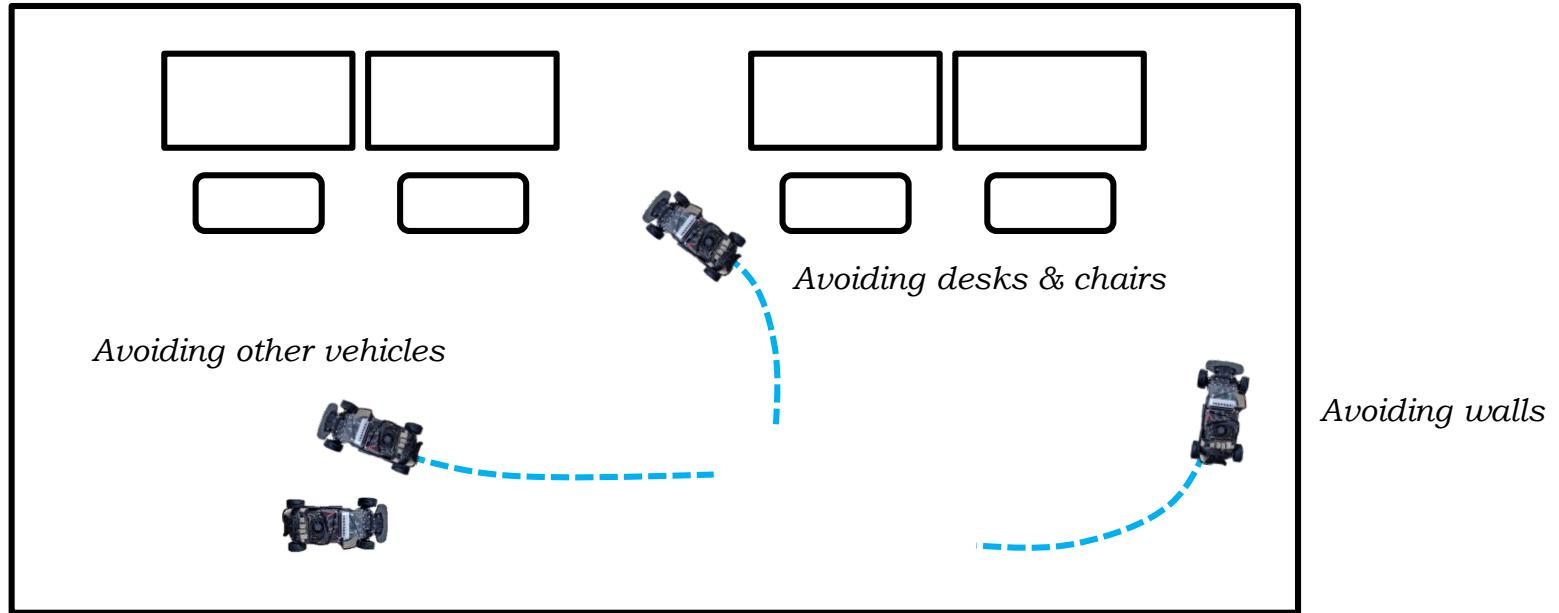
# Final Project Scenario

- ❖ Exploring an indoor environment to find target objects

- **Main Tasks:**

- **Avoid collisions with other vehicles and walls (or desks)**

- Plan collision-free path considering other vehicles, wall, and desks



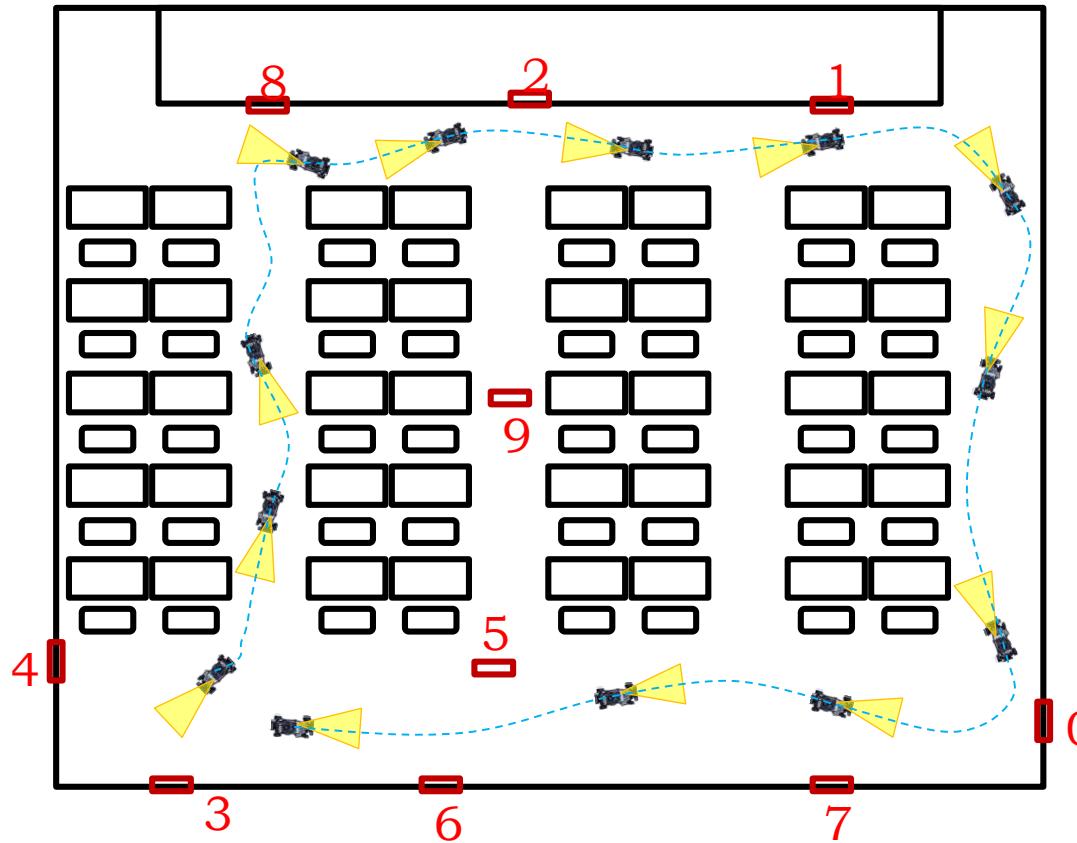
# Final Project Scenario

- ❖ Exploring an indoor environment to find target objects

- ❑ Main Tasks:

- Navigate the environment & Detect and Classify target objects

- Navigate the environment toward unexplored area without collision
      - Consider *backward maneuver* to avoid “stuck” situations



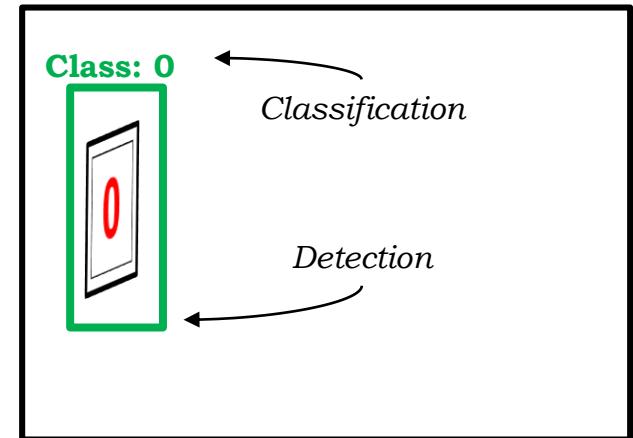
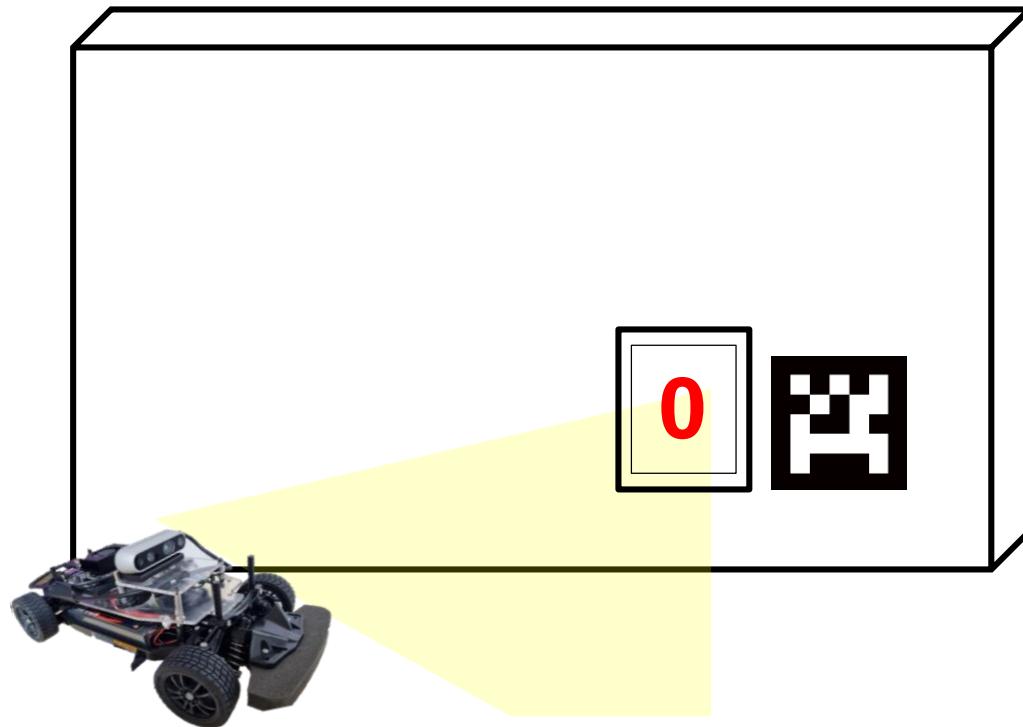
# Final Project Scenario

- ❖ Exploring an indoor environment to find target objects

- **Main Tasks:**

- Navigate the environment & Detect and Classify target objects

- Detect and classify target objects (using YOLO)
      - Save it as a file (to check it after finishing driving)



Front camera image  
(Realsense D435)

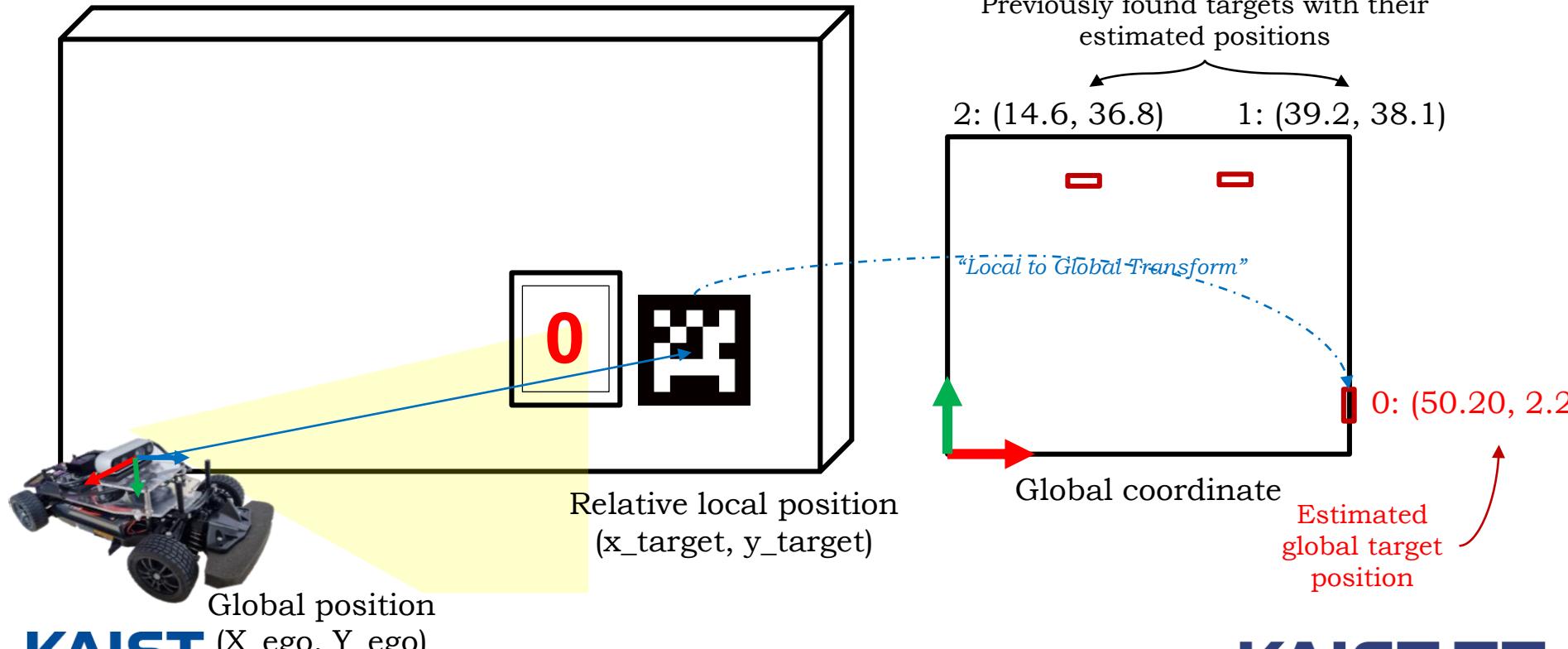
# Final Project Scenario

- ❖ Exploring an indoor environment to find target objects

- Main Tasks:

- Estimate the global position of the detected target object

- Calculate relative local position of the detected object
      - Transform the local position into the global position using ego's global pose



# Details

---

- ❖ Total 1 rounds
  - ❑ You will have 15 minutes
  - ❑ 2 opportunity to rescue your car if it stuck (-4pts)
  
- ❖ We will initialize following packages
  - ❑ Apriltag ROS
  - ❑ Apriltag localization
  - ❑ Detected image saver
  - ❑ SO DO NOT MODIFY OR ADD ANY CODES IN THESE PACKAGES!!**

# Location & Time

- ❖ Location: N1 room 117
- ❖ Time: December 8<sup>th</sup> 4pm ~



# N1 Room 117 Booking Status

## ❖ N1 room 117 booking status

- Greed – Booked for your preparation
- Red – Booked for other schedule (**!!Must clear the room!!**)



---

---

# Grading

# Grading

Homework	Attendance	Final	Final report
40	10	40	10

- ❖ Homework – 40% (8% \* 5)
- ❖ Final project result – 40%
  - Details are in the next page
- ❖ **Final report, presentation - 10%**
  - December 8<sup>th</sup>, after the final project
  - 5 minutes per team
  - Presentation of your team's work
    - Your approach for the exploration
    - Role of each team member

# Final Project Grading

---

- ❖ **Object detection – 4pts \* 10 objects**

- If you detect the object with correct classification, you earn 4 points.
  - If the classification is incorrect, -0.2 points.
  - Only the detection result in the “Detected image saver” will count.

- ❖ **(Additional points) Solve riddles using ChatGPT – 2pts \* 10 riddles**

# Solving Riddle using ChatGPT



ChatGPT

Certainly, here's a shorter version of the treasure hunt riddle:

In a room where time meets lore,  
Walk a hundred steps from the book-filled door.  
Turn right, count seats to five plus one,  
Beneath the sixth, the prize is won.



You

What is the answer of the riddle?



ChatGPT

The answer to the shortened riddle can be interpreted as follows:

1. **In a room where time meets lore:** This suggests a room that contains books and possibly a clock or a sense of history. It could be a library or a room with a significant collection of books.
2. **Walk a hundred steps from the book-filled door:** Start at the entrance to this room (presumably



You

Can you answer with one-line answer that can robot task planner understand?



ChatGPT

"Navigate to bookshelf, move 100 steps forward, turn right, locate sixth chair, check beneath."

---

---

# Detected Image Saver Node

# Detected Image Saver Node

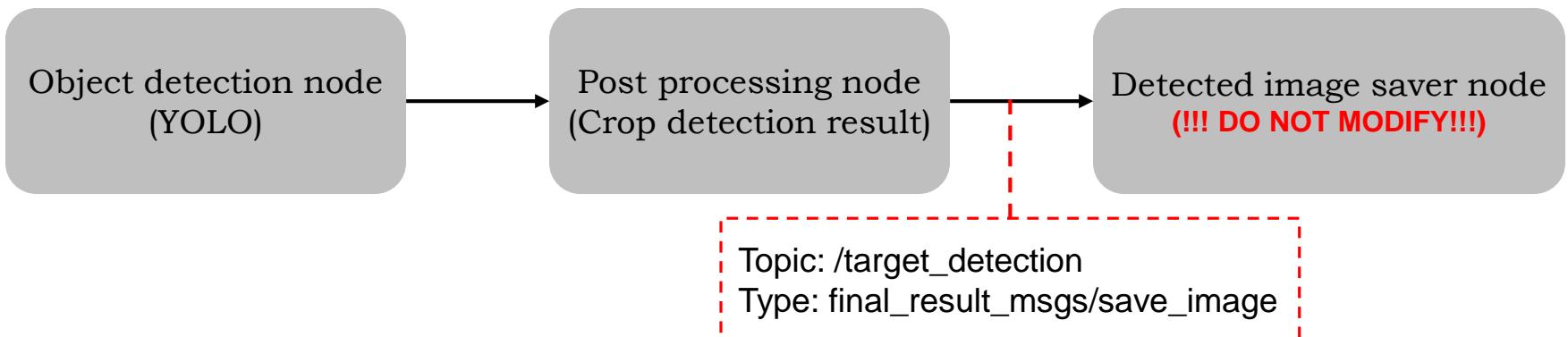
- ❖ Save Result for Final Race

- ❑ Class ID

- Find class ID (0~9)
    - Publish the class ID when you find it

- ❑ Image from Intel Realsense D435i

- Target object detection (Class ID: 0~9)
    - Crop the image within your YOLO bounding box
    - Publish the image of the class ID



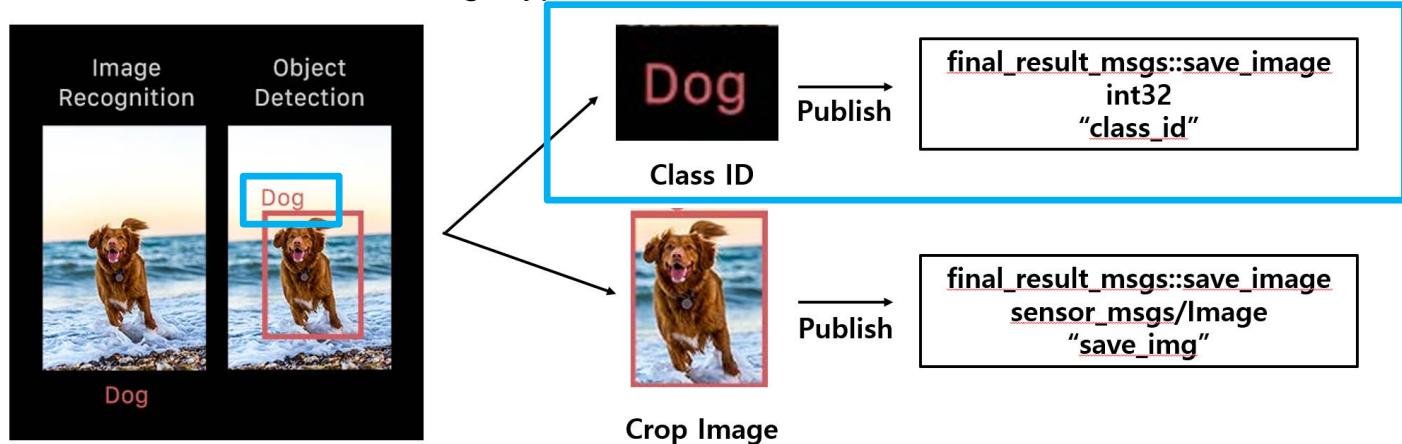
# Installation

---

- Two ROS package (Week14)
  1. final\_result
  2. final\_result\_msgs
- Download the two package
  - \$ cd (your workspace)/src  
ex) cd catkin\_ws/src
  - \$ git clone [https://github.com/HJKIMRobotics/detected\\_image\\_saver](https://github.com/HJKIMRobotics/detected_image_saver)
  - \$ cd ..
  - \$ catkin\_make

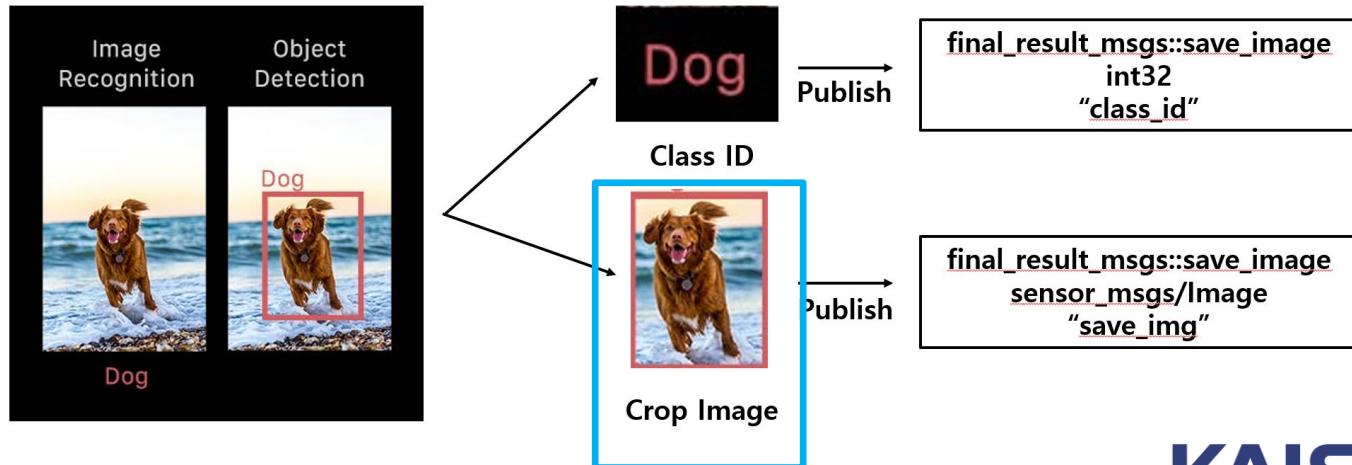
# Detected Image Saver Node

- 1. final\_result (**!!! IMPORTANT !!!**)
  - This node has two types of results ("target\_class\_id", "cv\_ptr", final\_x, final\_y)
  - Design your YOLO model to publish the **class ID, image, x\_pose, y\_pose**
  - Publish "Class" from your YOLO model to this node
  - Crop your final image to the size of the detected bounding box
  - Publish "Image" from your YOLO model to this node
  - Publish "x\_pose" and "y\_pose" from AprilTag
  - Ensure the publisher topic name is "/target\_detection"
- 2. final\_result\_msgs ("save\_image.msg") (**!!! DO NOT MODIFY THIS PACKAGE !!!**)
  - This package is a ROS custom message
  - It comprises two components, which are "int32 class\_id" and "sensor\_msgs/Image save\_img", respectively
  - After building this package, you must include this message in your YOLO model
    - ex) `#include "final_result_msgs/save_image.h"`
  - Please be careful to adhere to the message type



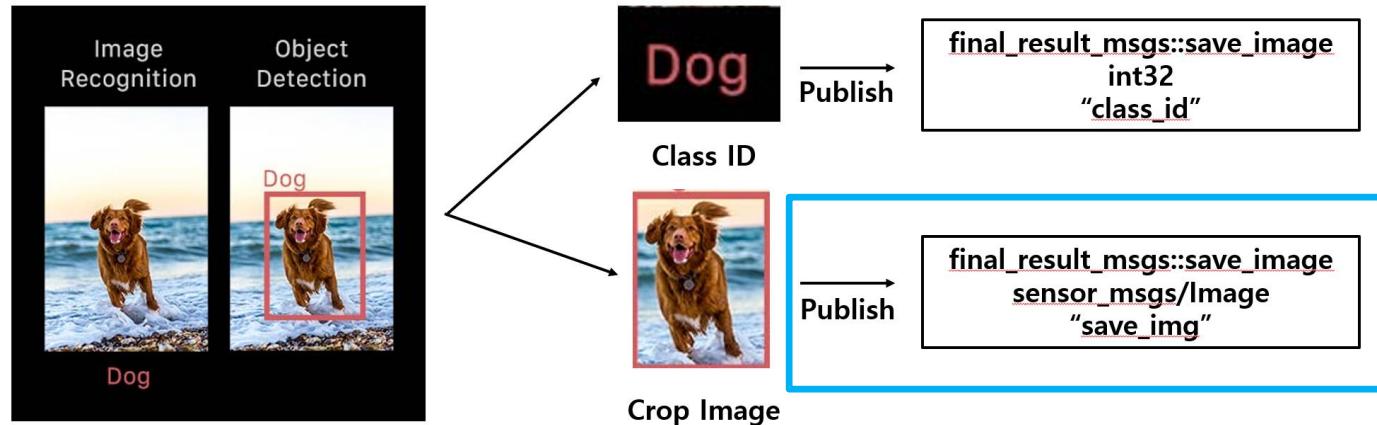
# Detected Image Saver Node

- 1. final\_result (**!!! IMPORTANT !!!**)
  - This node has two types of results ("target\_class\_id", "cv\_ptr", final\_x, final\_y)
  - Design your YOLO model to publish the **class ID, image, x\_pose, y\_pose**
  - Publish "Class" from your YOLO model to this node
  - **Crop your final image to the size of the detected bounding box**
  - Publish "Image" from your YOLO model to this node
  - Publish "x\_pose" and "y\_pose" from AprilTag
  - Ensure the publisher topic name is "/target\_detection"
- 2. final\_result\_msgs ("save\_image.msg") (**!!! DO NOT MODIFY THIS PACKAGE !!!**)
  - This package is a ROS custom message
  - It comprises two components, which are "int32 class\_id" and "sensor\_msgs/Image save\_img", respectively
  - After building this package, you must include this message in your YOLO model
    - ex) `#include "final_result_msgs/save_image.h"`
  - Please be careful to adhere to the message type



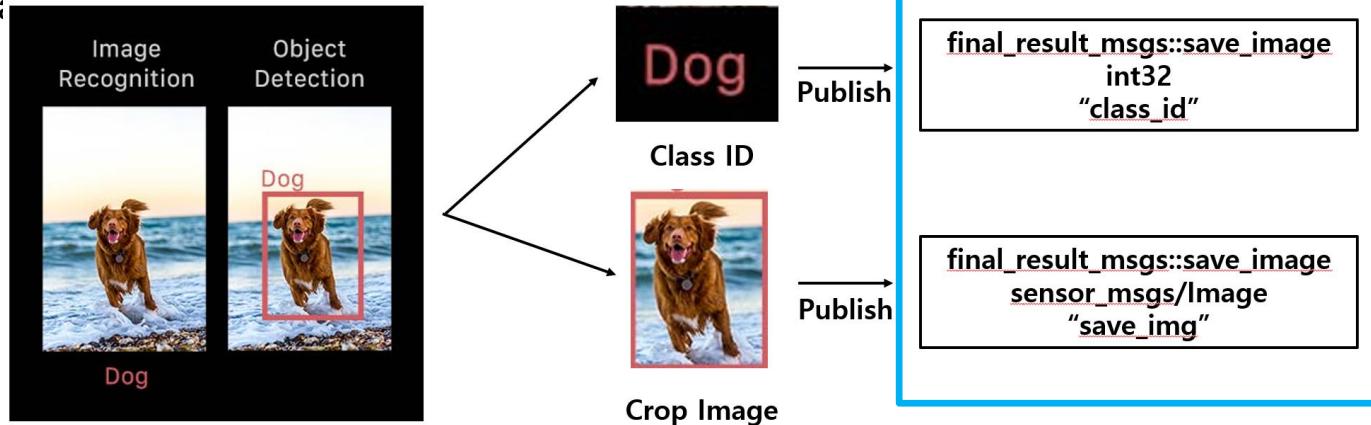
# Detected Image Saver Node

- 1. final\_result (**!!! IMPORTANT !!!**)
  - This node has two types of results ("target\_class\_id", "cv\_ptr", final\_x, final\_y)
  - Design your YOLO model to publish the **class ID, image, x\_pose, y\_pose**
  - Publish "Class" from your YOLO model to this node
  - Crop your final image to the size of the detected bounding box
  - Publish "Image" from your YOLO model to this node
  - Publish "x\_pose" and "y\_pose" from AprilTag
  - Ensure the publisher topic name is "/target\_detection"
- 2. final\_result\_msgs ("save\_image.msg") (**!!! DO NOT MODIFY THIS PACKAGE !!!**)
  - This package is a ROS custom message
  - It comprises two components, which are "int32 class\_id" and "sensor\_msgs/Image save\_img", respectively
  - After building this package, you must include this message in your YOLO model
    - ex) `#include "final_result_msgs/save_image.h"`
  - Please be careful to adhere to the message type



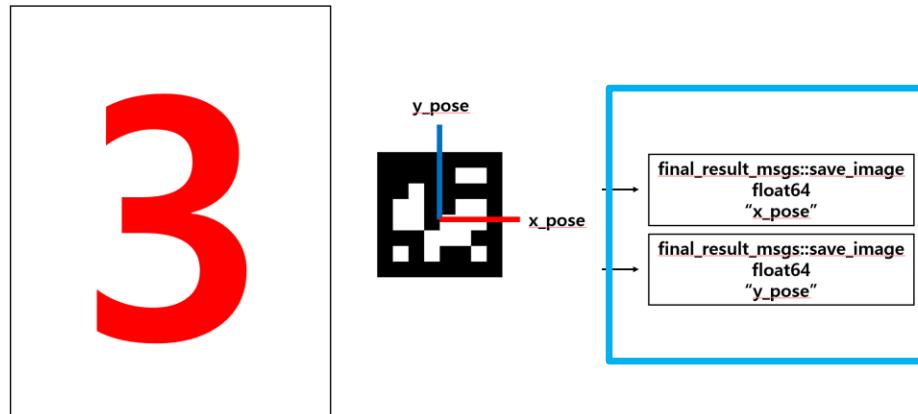
# Detected Image Saver Node

- 1. final\_result (**!!! IMPORTANT !!!**)
  - This node has two types of results ("target\_class\_id", "cv\_ptr", final\_x, final\_y)
  - Design your YOLO model to publish the **class ID, image, x\_pose, y\_pose**
  - Publish "Class" from your YOLO model to this node
  - Crop your final image to the size of the detected bounding box
  - Publish "Image" from your YOLO model to this node
  - Publish "x\_pose" and "y\_pose" from AprilTag
  - Ensure the publisher topic name is "/target\_detection"
- 2. final\_result\_msgs ("save\_image.msg") (**!!! DO NOT MODIFY THIS PACKAGE !!!**)
  - This package is a ROS custom message
  - It comprises four components, which are "int32 class\_id", "sensor\_msgs/Image save\_img", "float64 x\_pose", "float64 y\_pose", respectively
  - After building this package, you must include this message in your YOLO model
    - ex) #include "final\_result\_msgs/save\_image.h"
  - Please be careful about the file path.



# Detected Image Saver Node

- 1. final\_result (**!!! IMPORTANT !!!**)
  - This node has two types of results (“target\_class\_id”, “cv\_ptr”, final\_x, final\_y)
  - Design your YOLO model to publish the **class ID, image, x\_pose, y\_pose**
  - Publish “Class” from your YOLO model to this node
  - Crop your final image to the size of the detected bounding box
  - Publish “Image” from your YOLO model to this node
  - Publish “x\_pose” and “y\_pose” from AprilTag
  - Ensure the publisher topic name is “/target\_detection”
- 2. final\_result\_msgs (“save\_image.msg”) (**!!! DO NOT MODIFY THIS PACKAGE !!!**)
  - This package is a ROS custom message
  - It comprises four components, which are “int32 class\_id”, “sensor\_msgs/Image save\_img”, “float64 x\_pose”, “float64 y\_pose”, respectively
    - After building this package, you must include this message in your YOLO model  
*ex) #include “final\_result\_msgs/save\_image.h”*
    - Please be careful to the message type



# Detected Image Saver Node

- Save Result for Final Race
  - All student must be set their computer's local time to be the same
  - Automatically save a "jpg" file when you find numbers during the final race and publish the topics properly as mentioned in the previous slide.
    - When you run the "final\_result\_node" for the first time, an image will be automatically saved as a ".jpg" file ("jpg" is just used for testing the saving process)
    - The file name will be "[class\_id]\_[x\_pose(2 decimal)]\_[y\_pose(2 decimal)].jpg"
    - The final result might consist of up to 10 images if you find all the class ID during the race
- TA
  - Verify all your existing file
    - ex) 0.jpg, 1.jpg, 2.jpg, .... , 9.jpg
  - Compare the detected class ID from the image with the class ID from the file name
  - Evaluate the precision of the relative pose between your RC car and the AprilTag based on the x and y coordinates
  - Check the local time (hour, minute, second) at the moment the file was saved

---

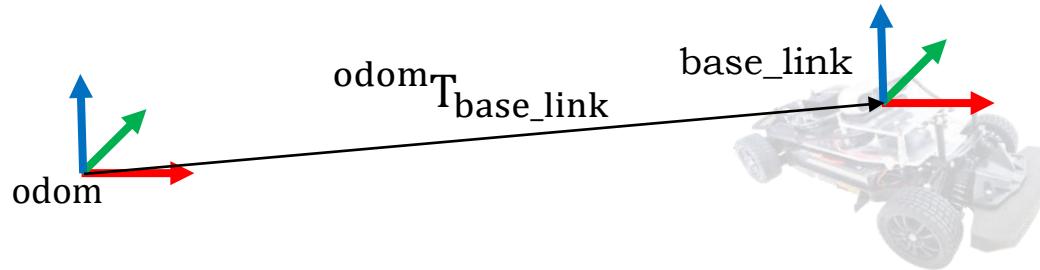
---

# AprilTags Localization (GT Guidance)

# Localization

## ❖ ORB SLAM

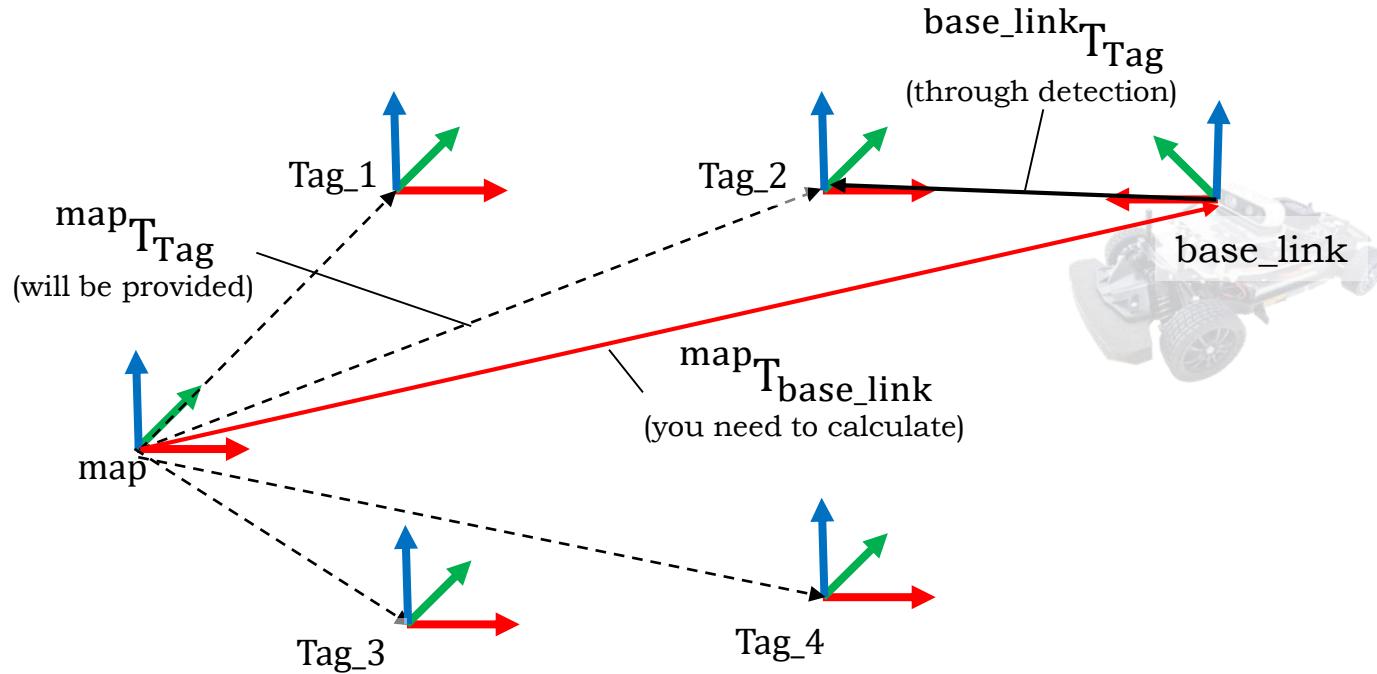
- ❑ “odom” frame → “base\_link” of the car
- ❑ The origin differs by where you start the algorithm.  
(starting point will be the origin of the “odom” frame)
- ❑ It might not be accurate after the car makes rapid turn  
(due to low frame rate and feature loss)



# AprilTags Localization

## ❖ AprilTag localization

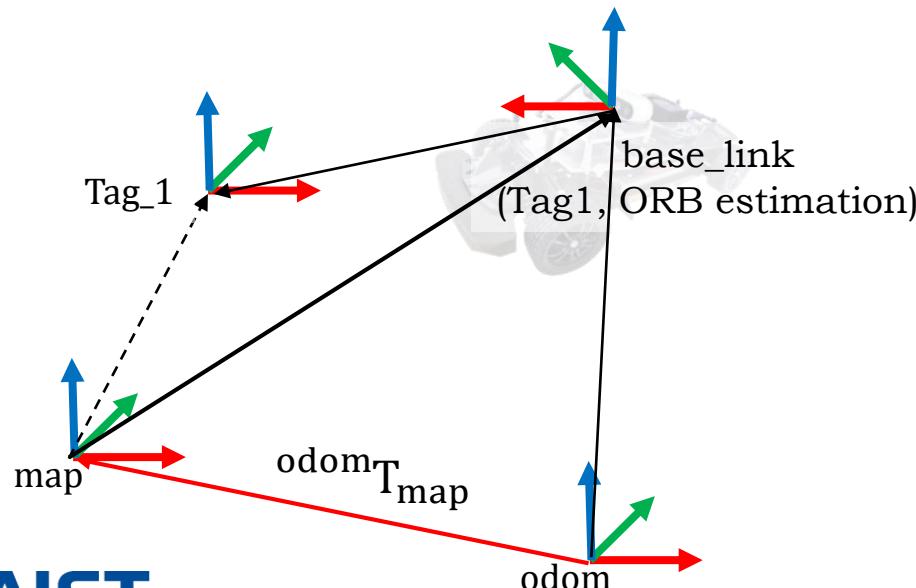
- Based on fixed “map” frame
- Provide very accurate pose of the detected tag
- Only available when tag is detected



# AprilTags Fusion Localization

- ❖ Global map alignment (Initially – car will be placed in front of a Tag)
  - When Tag and ORB is both available
  - Use both localizer's result to initialize transformation between “map” and “odom” frame ( ${}^{\text{odom}}\mathbf{T}_{\text{map}}$ )

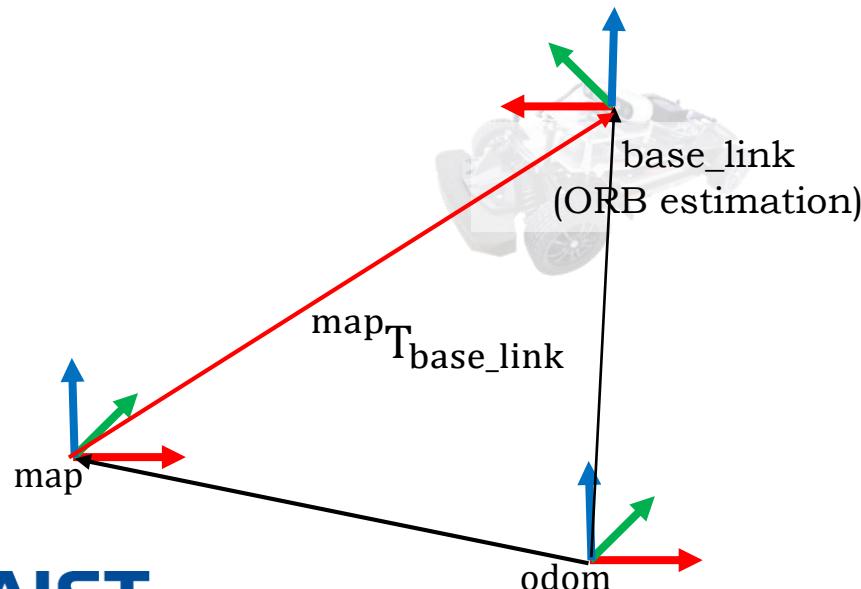
$${}^{\text{odom}}\mathbf{T}_{\text{map}} = {}^{\text{odom}}\mathbf{T}_{\text{base\_link}} \frac{{}^{\text{map}}\mathbf{T}_{\text{base\_link}}^{-1}}{\text{ORB SLAM} \quad \text{Tag localization}}$$



# AprilTags Fusion Localization

- ❖ Fusion localization (When tags are out of sight)
  - You need to rely on ORB SLAM for the localization
  - Use  ${}^{\text{odom}}\mathbf{T}_{\text{map}}$  and ORB SLAM result to localize the “base\_link” on the “map” frame

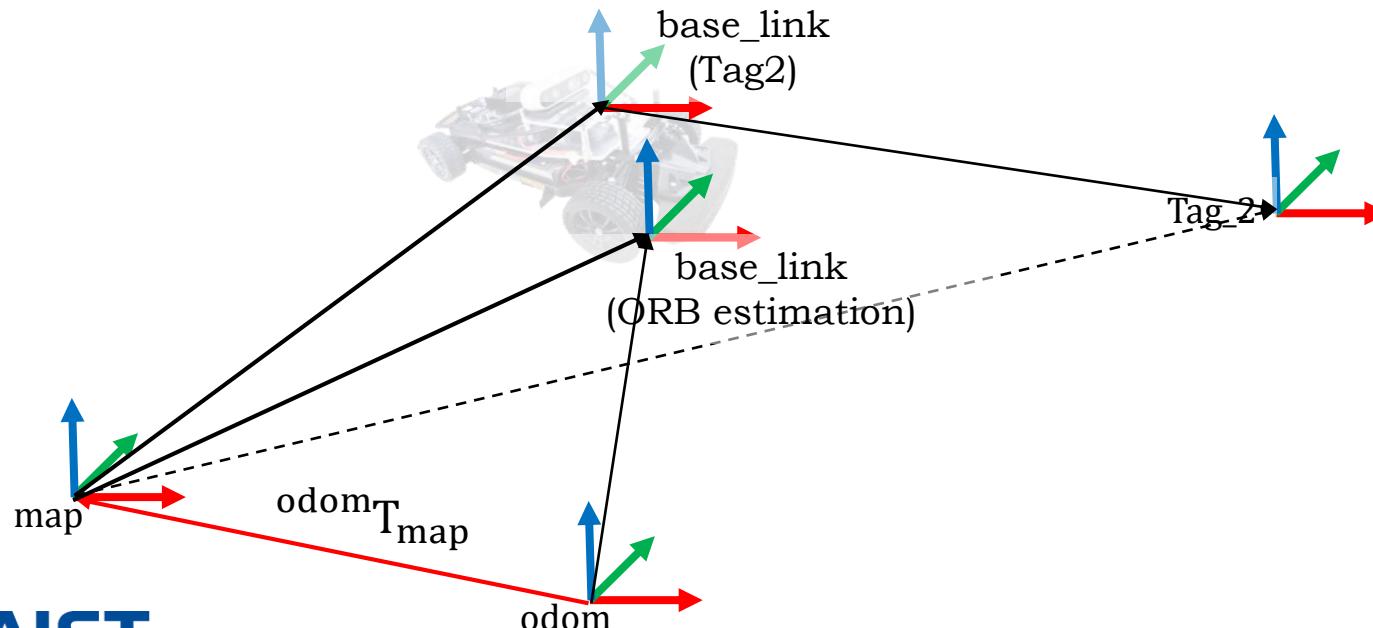
$${}^{\text{map}}\mathbf{T}_{\text{base\_link}} = \frac{{}^{\text{map}}\mathbf{T}_{\text{odom}}}{{}^{\text{Global map alignment}}} {}^{\text{odom}}\mathbf{T}_{\text{base\_link}} \quad \text{ORB SLAM}$$



# AprilTags Fusion Localization

- ❖ Global map realignment (When tags are in sight)
  - ❑ There will be error in ORB SLAM result ( ${}^{\text{map}}\mathbf{T}_{\text{base\_link}}$ )
  - ❑ Recalculate the  ${}^{\text{odom}}\mathbf{T}_{\text{map}}$  using the AprilTags localization and ORB SLAM localization

$${}^{\text{odom}}\mathbf{T}_{\text{map}} = {}^{\text{odom}}\mathbf{T}_{\text{base\_link}} \frac{{}^{\text{map}}\mathbf{T}_{\text{base\_link}}^{-1}}{\text{ORB SLAM} \quad \text{Tag localization}}$$

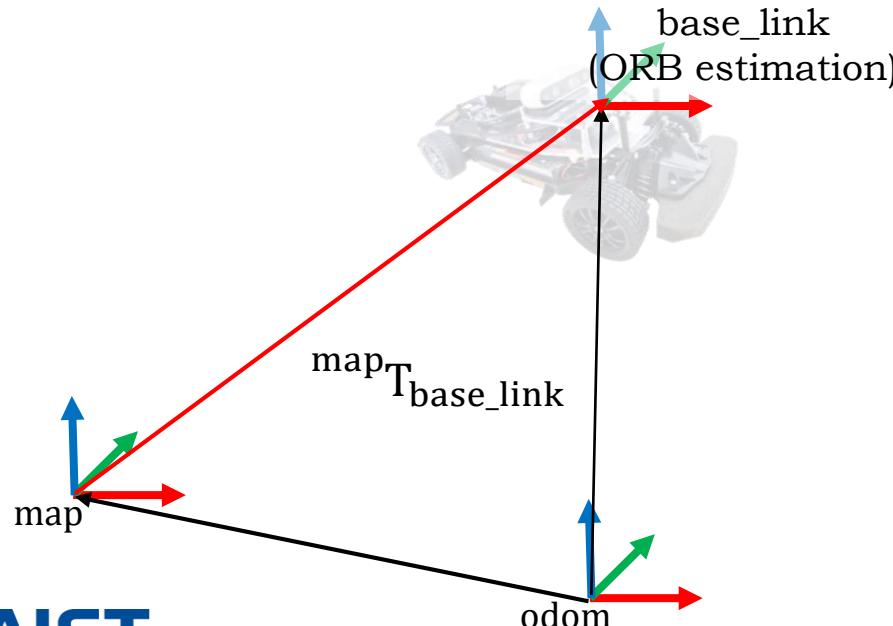


# AprilTags Fusion Localization

- ❖ Fusion localization (When tags are out of sight)
  - You need to rely on ORB SLAM for the localization
  - Use recalculated  ${}^{\text{odom}}\mathbf{T}_{\text{map}}$  and ORB SLAM result to localize the “base\_link” on the “map” frame

$${}^{\text{map}}\mathbf{T}_{\text{base\_link}} = \frac{{}^{\text{map}}\mathbf{T}_{\text{odom}}}{{}^{\text{odom}}\mathbf{T}_{\text{base\_link}}}$$

Global map realignment      ORB SLAM

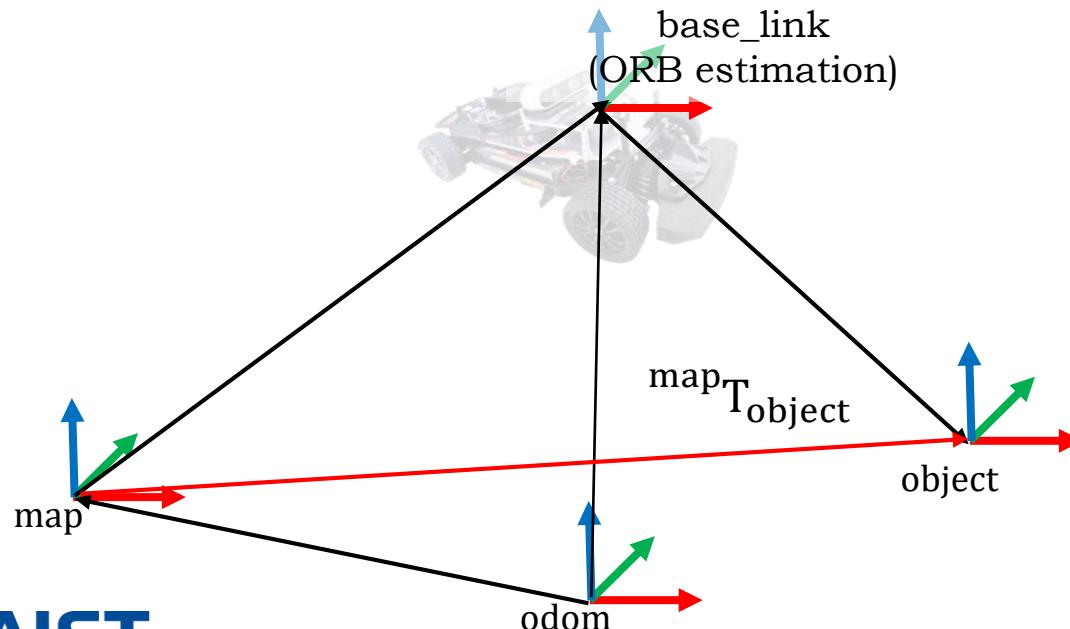


# AprilTags Fusion Localization

- ❖ Object position estimation

- ❑ There will be tags attached next to the target
- ❑ You need to transform the target object's pose to “map” frame to estimate it's global position

$$\text{map}_T_{\text{object}} = \frac{\text{map}_T_{\text{base\_link}}}{\text{Localization result}} \frac{\text{base\_link}_T_{\text{object}}}{\text{AprilTag detection}}$$



# AprilTags Localization

## ❖ What are AprilTags?

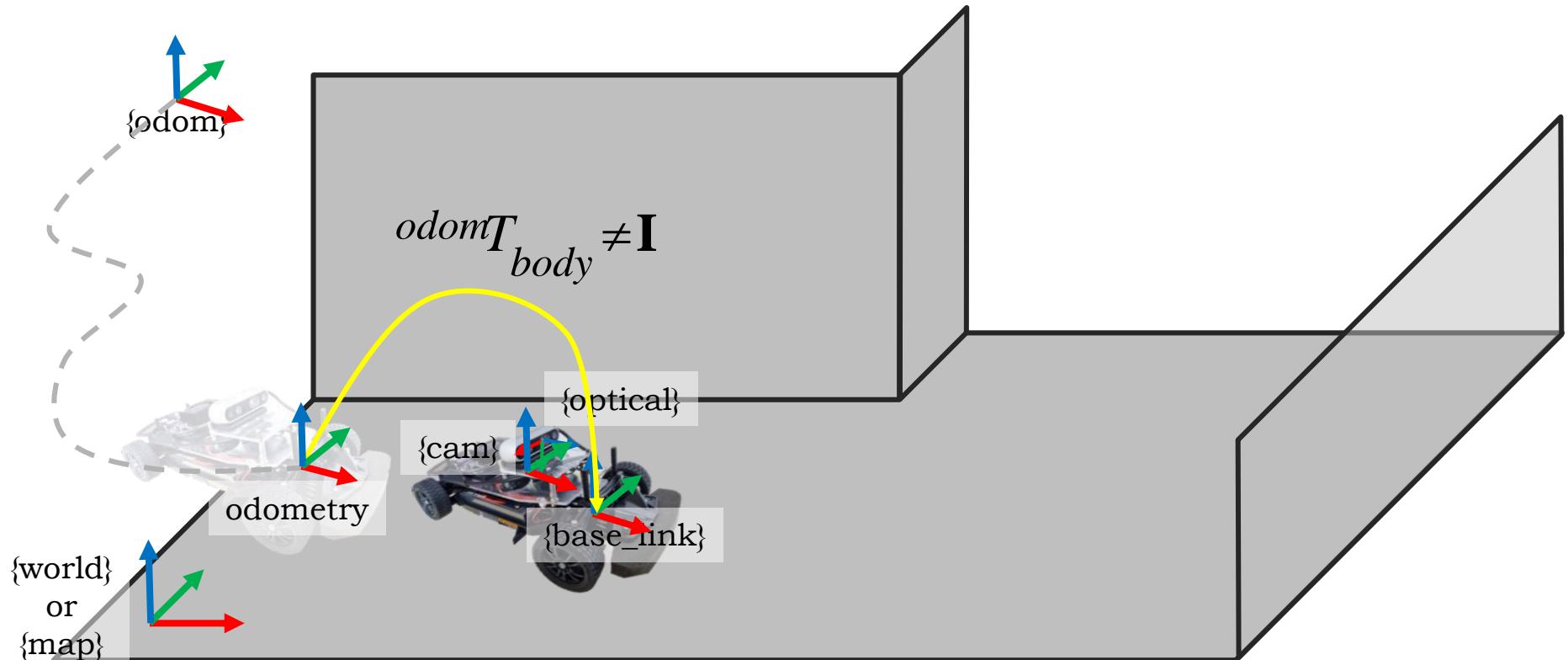
□ **AprilTags** is a visual fiducial system, useful for a wide variety of tasks including augmented reality, robotics, and camera calibration.

- Can be created from an ordinary printer
- Detection software computes the precise 3D position, orientation, and identity of the tags relative to the camera
- Library is implemented in C with no external dependencies
- Designed to be easily included in other applications
- Real-time performance can be achieved even on cell-phone grade processors.



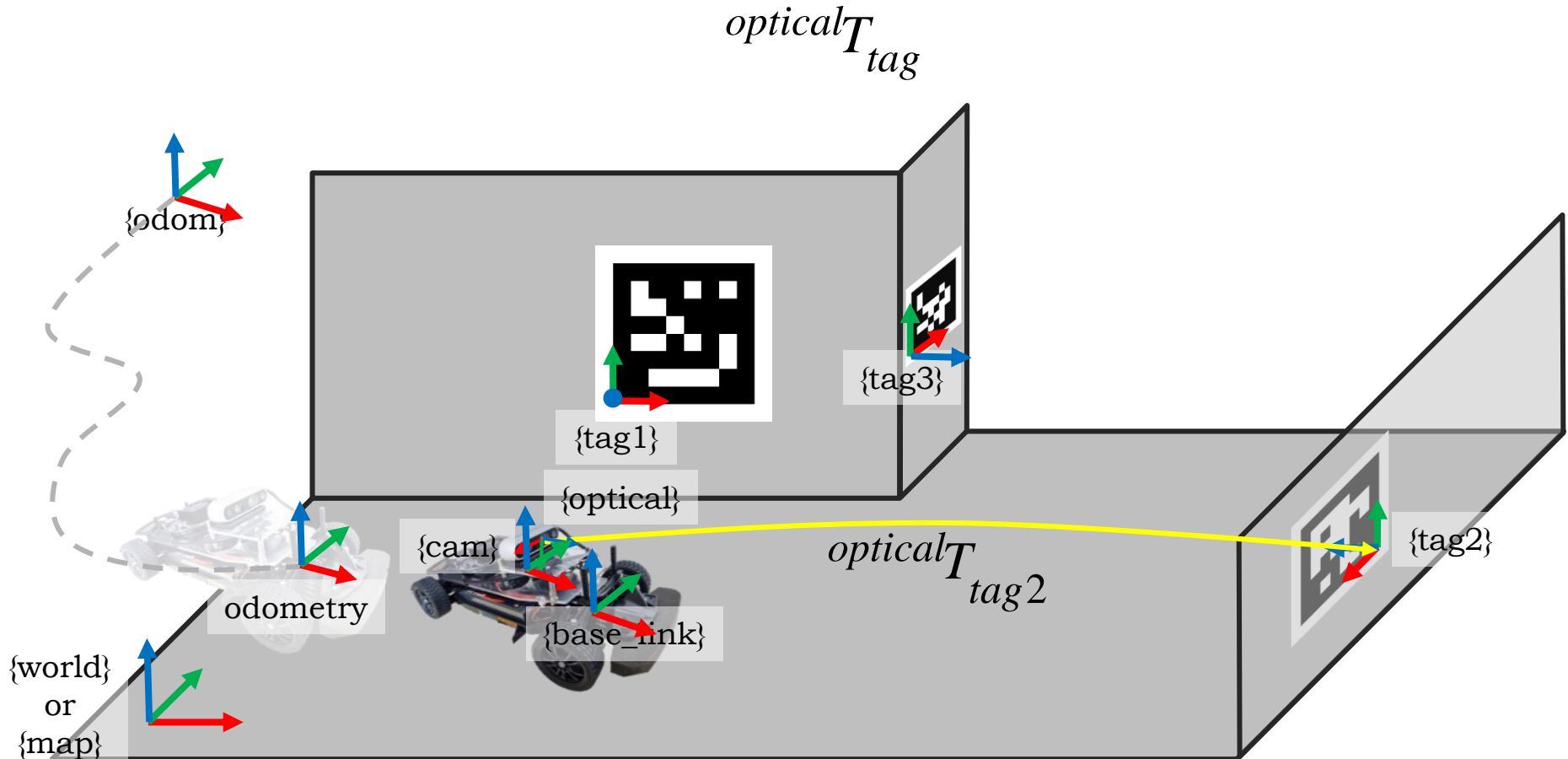
# AprilTags Localization

- ❖ Why use AprilTags localization?
  - ❑ SLAM algorithm (in our case, ORB) odometry error accumulates while move around the world and generating map.
    - **We have to correct localization errors!**



# AprilTags Localization

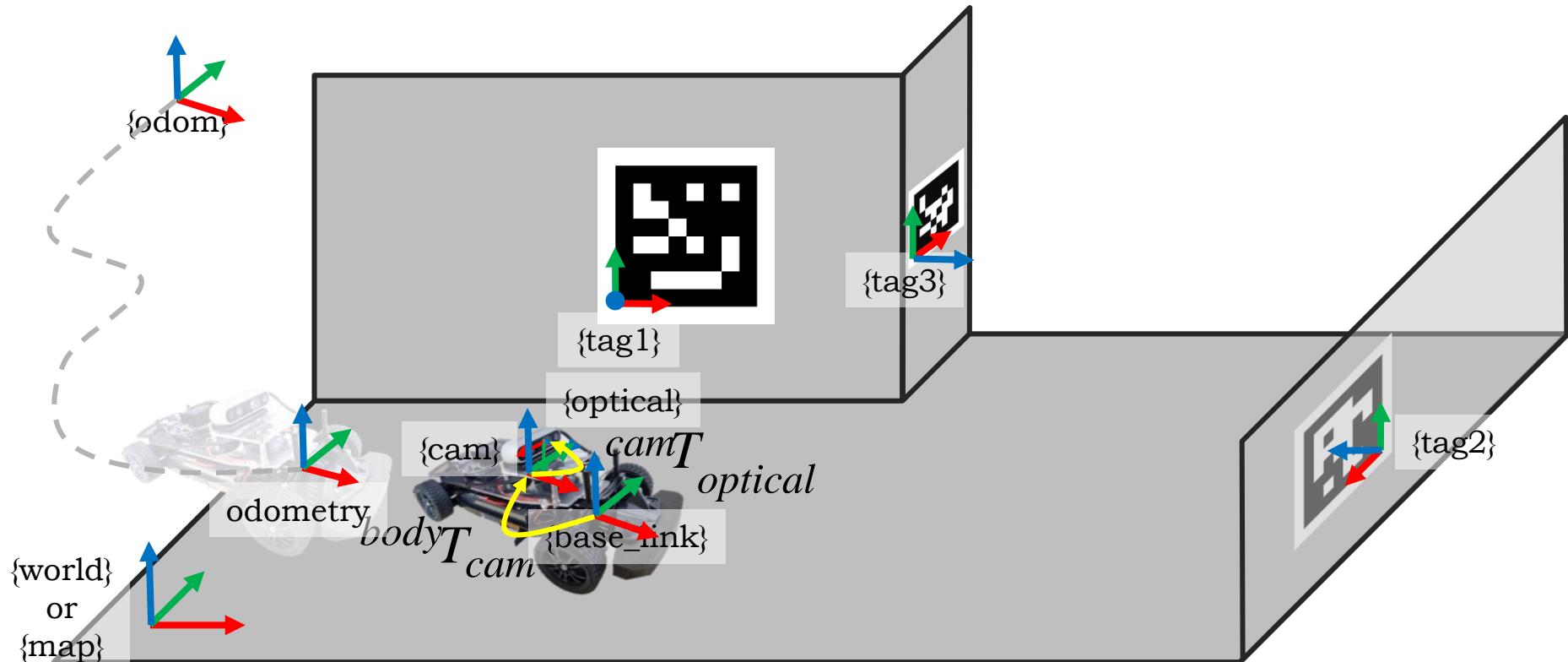
- ❖ How to use AprilTags in localization problem?
  - If you know the size of AprilTags, then you can find transformation between camera's optical frame and tag frame.



# AprilTags Localization

- ❖ How to use AprilTags in localization problem?
  - Also we know transformation robot body to cam, cam to optical.

$$bodyT_{cam} \quad camT_{optical}$$



\* $\{camera\}$  is camera mount frame given from camera URDF

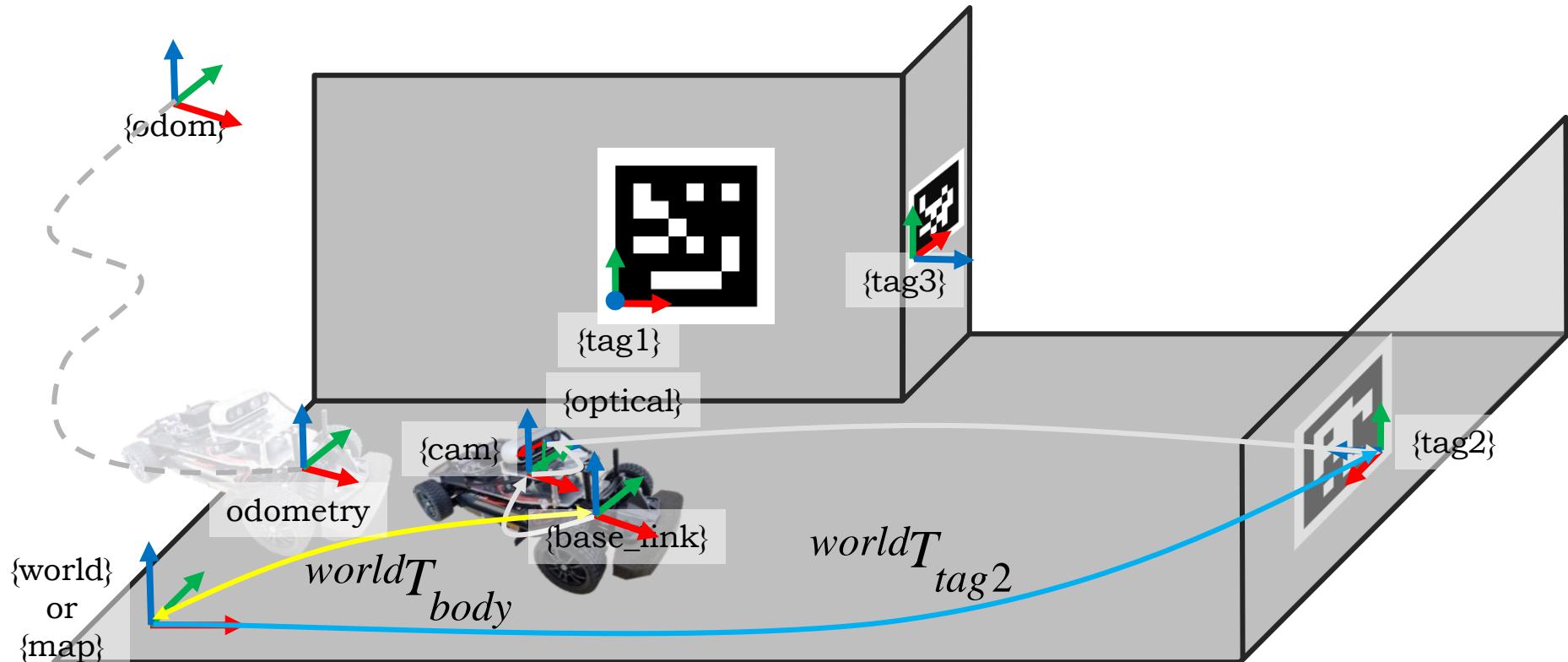
\*\* $\{optical\}$  is actual camera image sensor frame from camera URDF

EE 405A

# AprilTags Localization

- ❖ How to use AprilTags in localization problem?
  - If we know world(or map) frame to tag transformation, we can calculate TRUE robot's coordinate w.r.t. world frame.

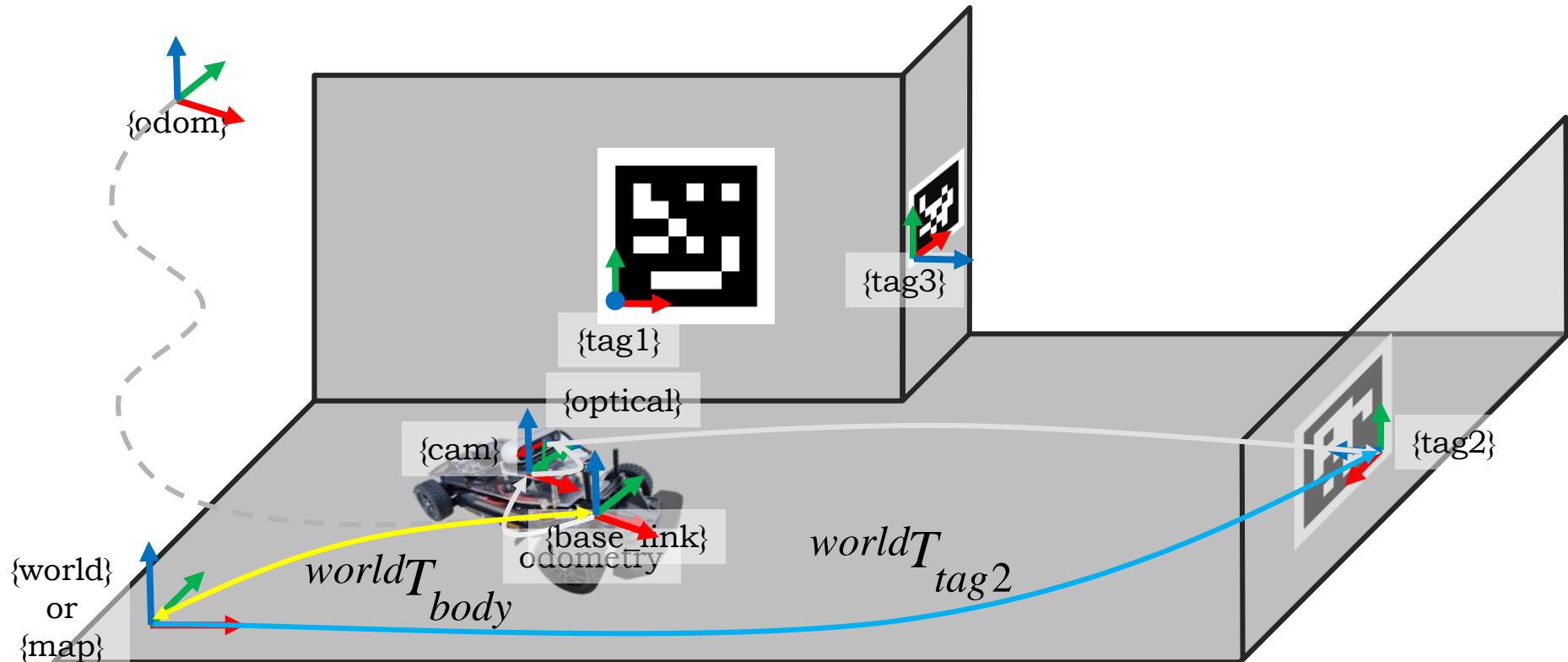
$$worldT_{body} = worldT_{tag} \cdot opticalT_{tag}^{-1} \cdot camT_{optical}^{-1} \cdot bodyT_{cam}^{-1}$$



# AprilTags Localization

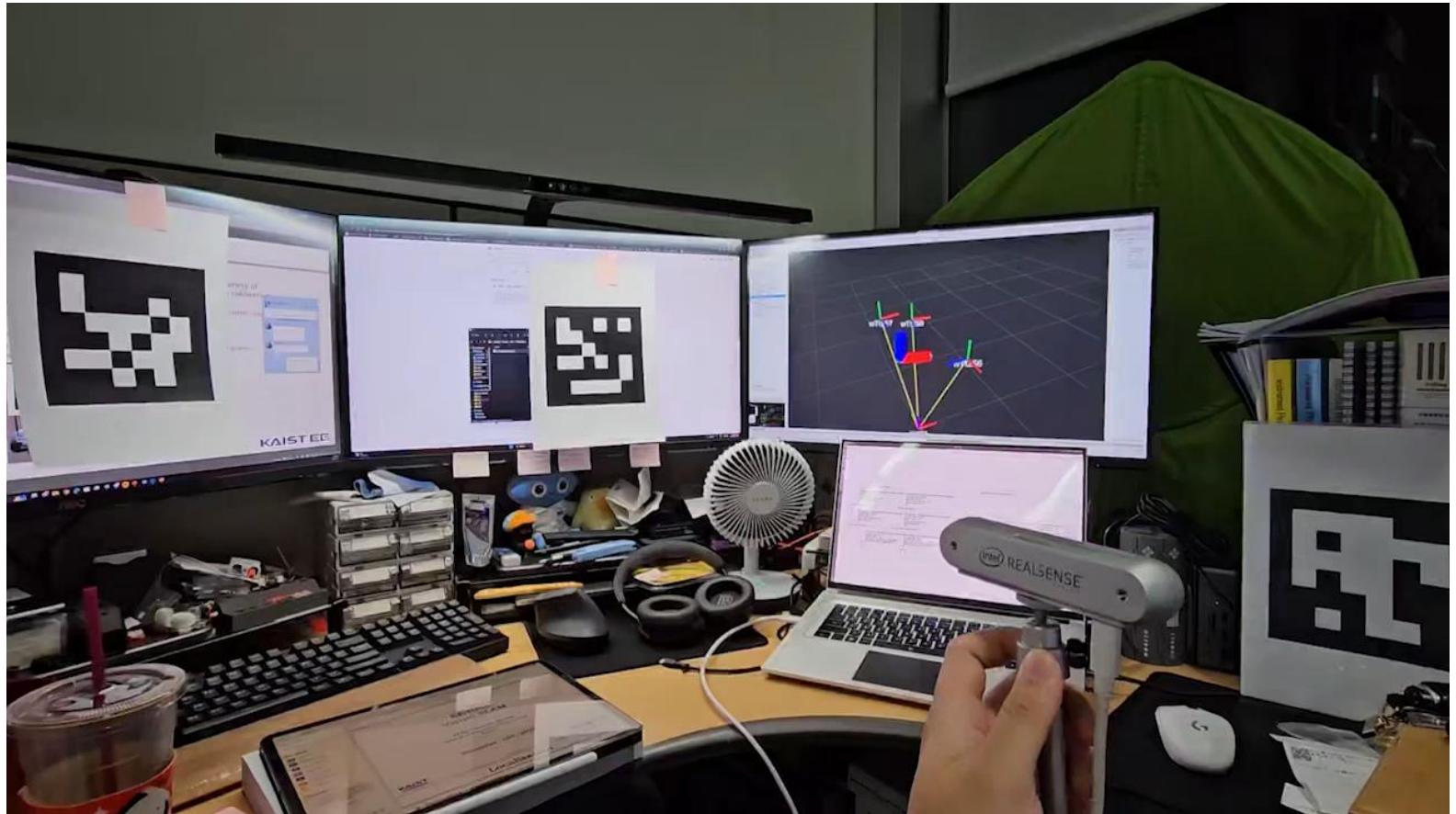
- ❖ How to use AprilTags in localization problem?
  - If we know world(or map) frame to tag transformation, we can calculate TRUE robot's coordinate w.r.t. world frame.

$$worldT_{body} = worldT_{tag} \cdot opticalT_{tag}^{-1} \cdot camT_{optical}^{-1} \cdot bodyT_{cam}^{-1}$$



# AprilTags Localization

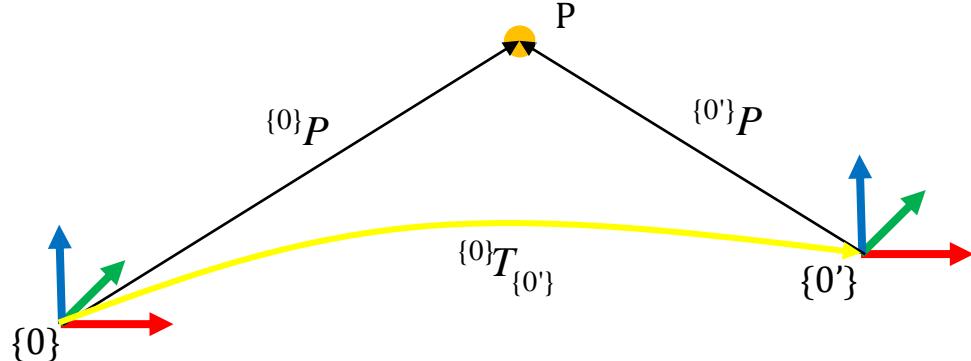
- ❖ True RT Tag localization node
  - Localizing camera in multi-AprilTags environment with closest Tag
    - True RT Tag node provide ground truth transformation of AprilTags w.r.t.  $worldT_{tag}$



# Transformation

## ❖ Translation transformation

- ❑ Frames, { }: reference, base, inertial, base, world, end-effector, object



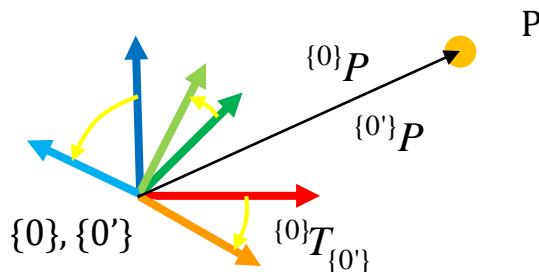
where  $\{0\}P = (x_0, y_0, z_0)$

$\{0'\}P = (x_1, y_1, z_1)$

translation

$$\{0\}P = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \{0\}T_{0'} \{0'\}P$$

## ❖ Rotation transformation



where  $\{0\}P = x_0\mathbf{i} + y_0\mathbf{j} + z_0\mathbf{k}$

$\{0'\}P = x_1\mathbf{i}' + y_1\mathbf{j}' + z_1\mathbf{k}'$

rotation  $x_0 = P \cdot \mathbf{i} = (x_1\mathbf{i}' + y_1\mathbf{j}' + z_1\mathbf{k}') \cdot \mathbf{i}$

$y_0 = P \cdot \mathbf{j} = (x_1\mathbf{i}' + y_1\mathbf{j}' + z_1\mathbf{k}') \cdot \mathbf{j}$

$z_0 = P \cdot \mathbf{k} = (x_1\mathbf{i}' + y_1\mathbf{j}' + z_1\mathbf{k}') \cdot \mathbf{k}$

matrix from

$$\{0\}P = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{i} \cdot \mathbf{i}' & \mathbf{i} \cdot \mathbf{j}' & \mathbf{i} \cdot \mathbf{k}' & 0 \\ \mathbf{j} \cdot \mathbf{i}' & \mathbf{j} \cdot \mathbf{j}' & \mathbf{j} \cdot \mathbf{k}' & 0 \\ \mathbf{k} \cdot \mathbf{i}' & \mathbf{k} \cdot \mathbf{j}' & \mathbf{k} \cdot \mathbf{k}' & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \{0\}T_{0'} \{0'\}P$$

## ❖ Homogeneous transformation matrix

- ❑ Can describe 6 DOF

$$T \triangleq \left[ \begin{array}{c|c} R_{3 \times 3} & P_{3 \times 1} \\ \hline 0_{1 \times 3} & 1 \end{array} \right]$$

# ROS TF2 Library

## ❖ TF2 Library?

- ❑ TF2 is a library for transformation which lets the users keep track multiple coordinate frames over time.

➤ <https://wiki.ros.org/tf2>

- ❑ Tutorials are available

➤ <https://wiki.ros.org/tf2/Tutorials>

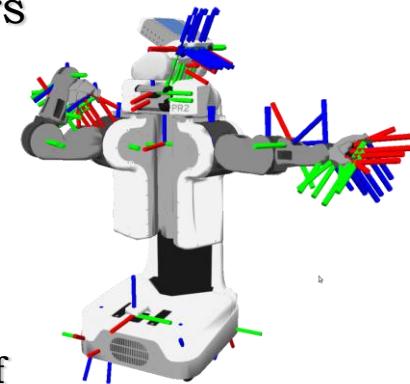
- Please follow basic broadcaster and **listener** tutorial.
- After following tutorial, please try to understand the code of True RT Tag package.
- It will be very helpful for implementing your final project.

- ❑ Support easy computation of transformations

$$\textit{worldT}_{cam} = \textit{worldT}_{tag} \cdot \textit{opticalT}_{tag}^{-1} \cdot \textit{camT}_{optical}^{-1}$$

||

```
tf2::Transform world2cam_tf = world2tag_tf*optical2tag_tf.inverse()*cam2optical_tf.inverse();
```



# ROS TF2 Library

## ❖ Basic functionalities of TF2

- ❑ tf2::TransformListener = Request and receive /TF
- ❑ tf2::TransformBroadcaster = Publish new transformation
  
- ❑ tf2::Transform = C++ class providing homogeneous transform
- ❑ tf2::Quaternion = C++ class providing rotational transformation
  - Quaternion.set... and Quaternion.get... methods are available to define and get values of rotation.
  - Quaternion can be multiplied to compute rotation only.

```
tf2::Quaternion x90(DEG2RAD(0.0), DEG2RAD(90.0), DEG2RAD(0.0));
tf2::Quaternion y_90(DEG2RAD(-90.0), DEG2RAD(00.0), DEG2RAD(0.0));
tf2::Quaternion orientation = x90*y_90;
```
  
- ❑ tf2::fromMsg(input GEOMETRY\_MSGS, output TF2\_DATA\_TYPES)
- ❑ tf2::toMsg(input TF2\_DATA\_TYPES, output GEOMETRY\_MSGS)
  - These functions are overloaded with multiple data types, so you can change ROS default data types into TF2 and compute.

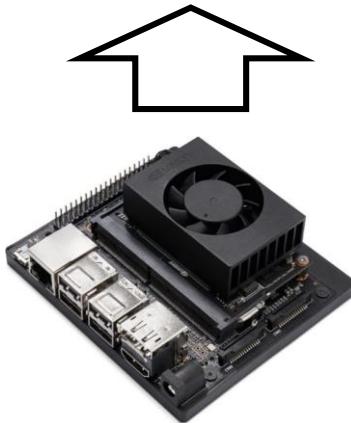
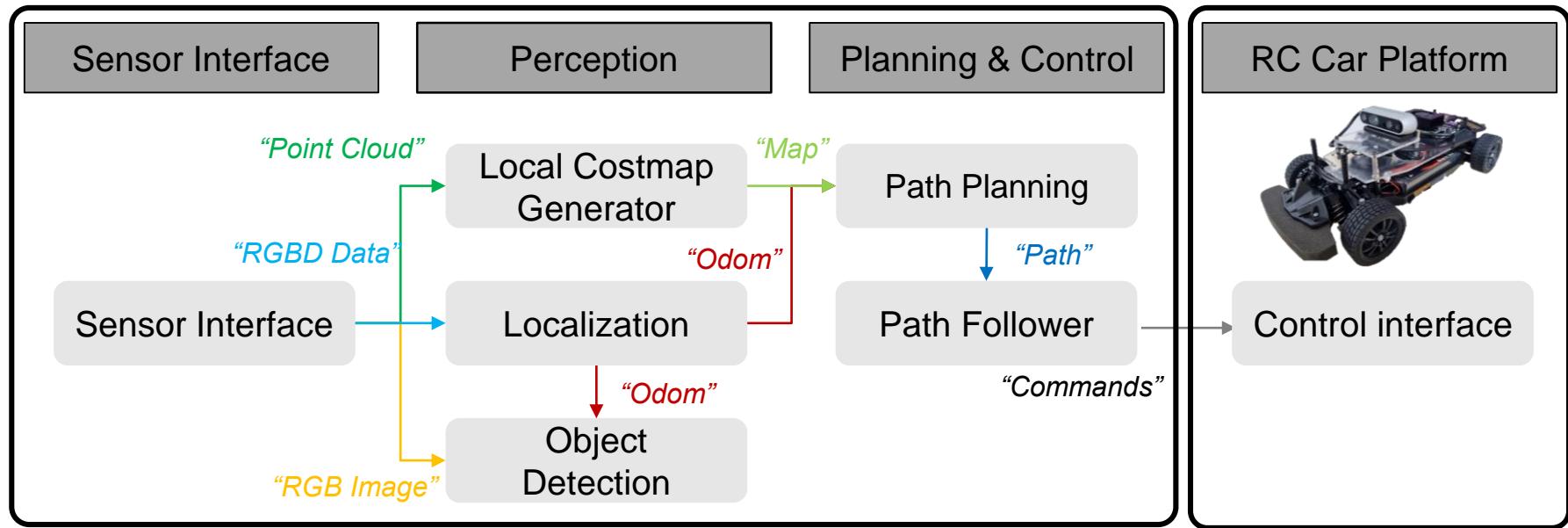
```
geometry_msgs::Pose cam2tag_pose = tag_detection.detections.at(min_idx).pose.pose.pose;
tf2::Transform optical2tag_tf;
tf2::fromMsg(cam2tag_pose, optical2tag_tf);
```

---

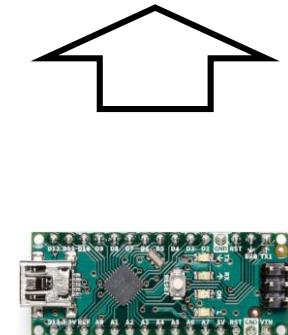
---

# Overall System Architecture

# Overall System Architecture

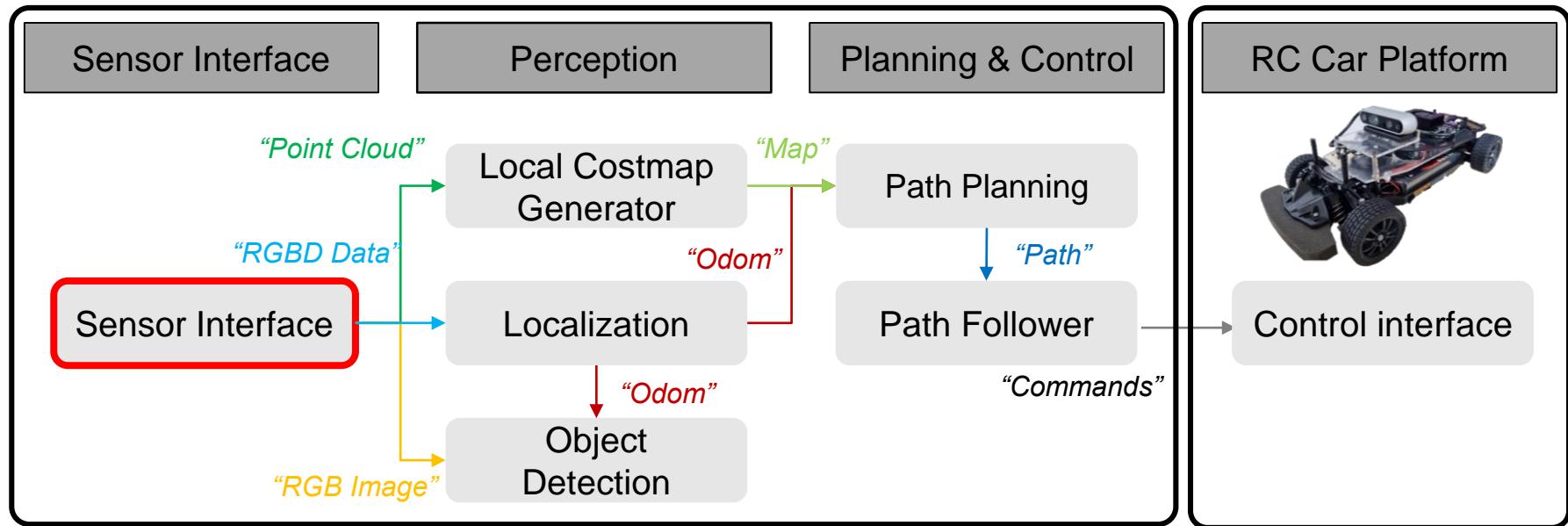


Jetson Xavier NX



Arduino Nano

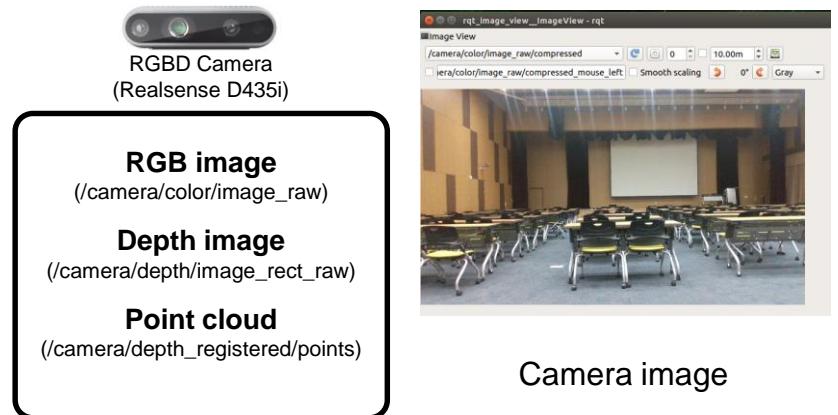
# Overall System Architecture



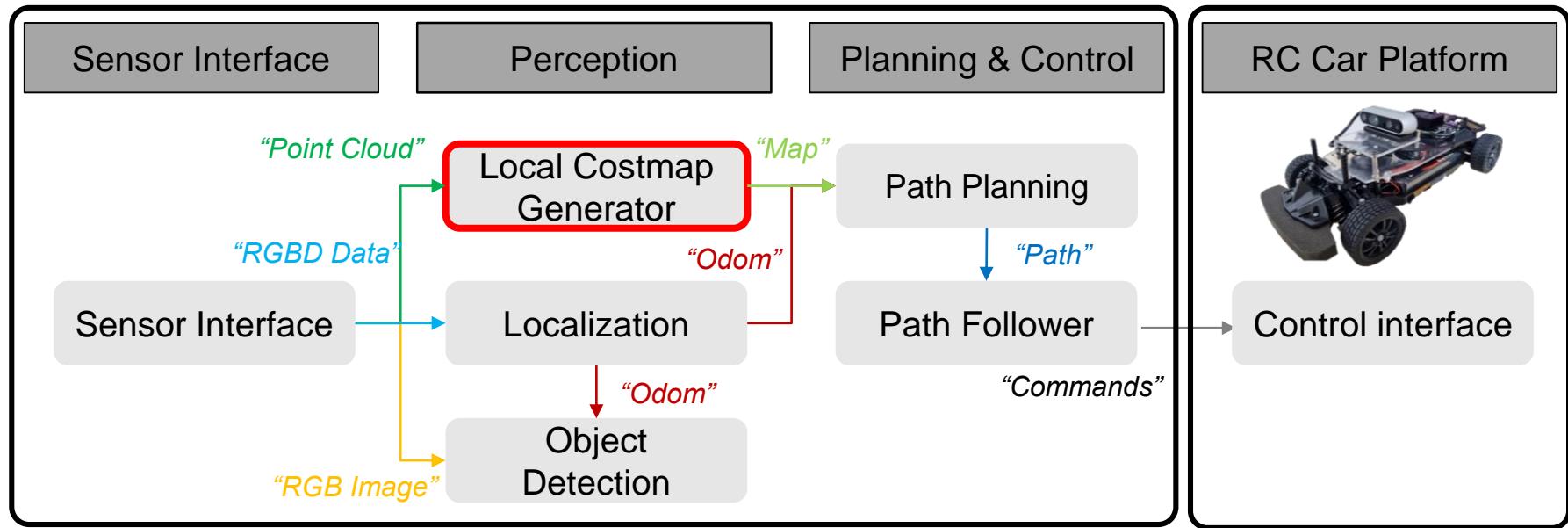
## ➤ Sensor Interface (realsense2\_camera)

- ❑ Receive sensor data, such as RGB image, depth image, Point cloud, from the Realsense D435 interface ROS package.
- ❑ You can utilize “depthImage2PointCloud” ROS package for point cloud generation from depth images, which is more lightweighted way than realsense package.

<https://github.com/Guri-cccc/depthImage2PointCloud.git>



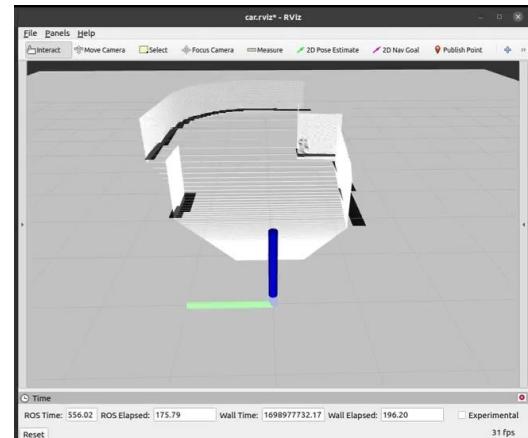
# Overall System Architecture



## ➤ Local Costmap Generator (`local_costmap_generator`)

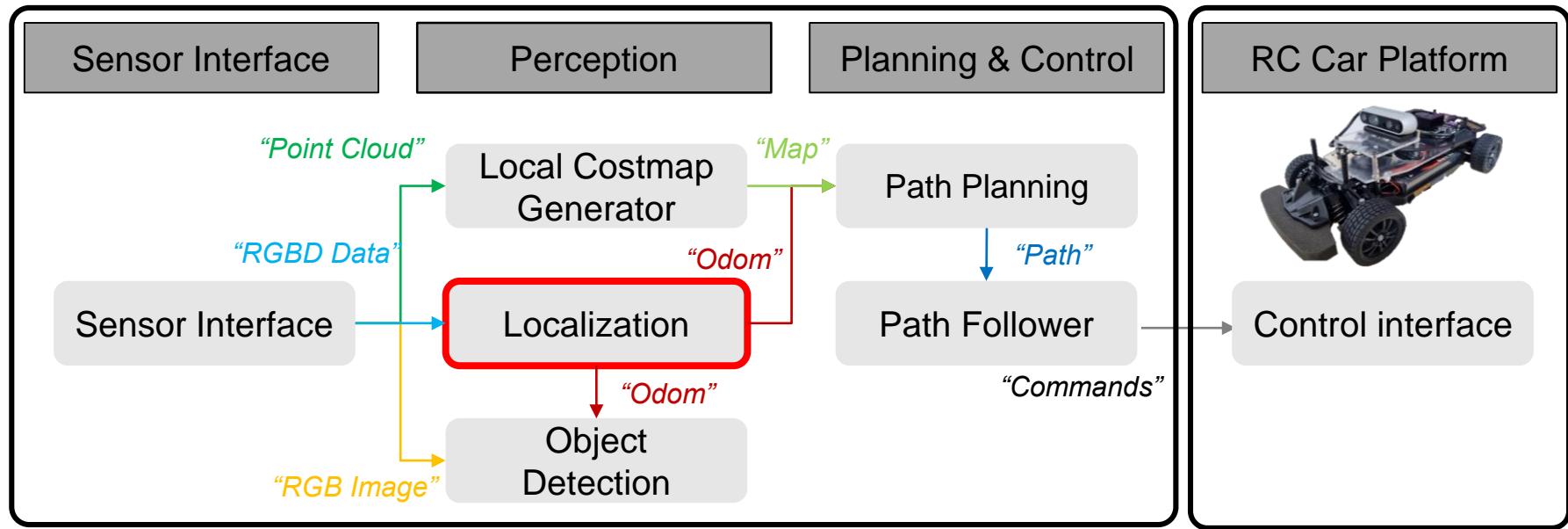
- ❑ The local costmap node generates occupancy grid maps based on the **point cloud data** from RGB-D camera sensor.
- ❑ **heightmap\_node**: filtering points in a specific height range.
- ❑ **heightmap\_costmap\_node**: transform the filtered point cloud to occupancy grid map.

[https://github.com/hynkis/local\\_costmap\\_generator](https://github.com/hynkis/local_costmap_generator)



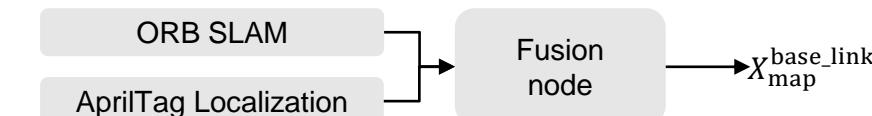
Height-based local cost map

# Overall System Architecture



## ➤ Localization (orb3\_ros\_interface & true\_rt\_tag)

- ❑ Get ego vehicle's odometry through visual SLAM
- ❑ Align odometry with global map using Tag localization



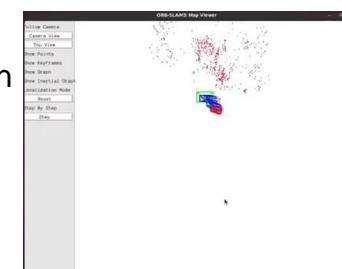
[ORB SLAM]

[https://github.com/yongeePark/ORB\\_SLAM3\\_ROS\\_Interface](https://github.com/yongeePark/ORB_SLAM3_ROS_Interface)

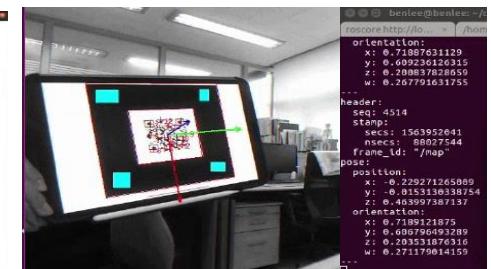
[AprilTag localization]

[https://github.com/MinSungjae/true\\_rt\\_tag/tree/123236c3d92a2e95319](https://github.com/MinSungjae/true_rt_tag/tree/123236c3d92a2e95319)

277fc8653411e7c609a16

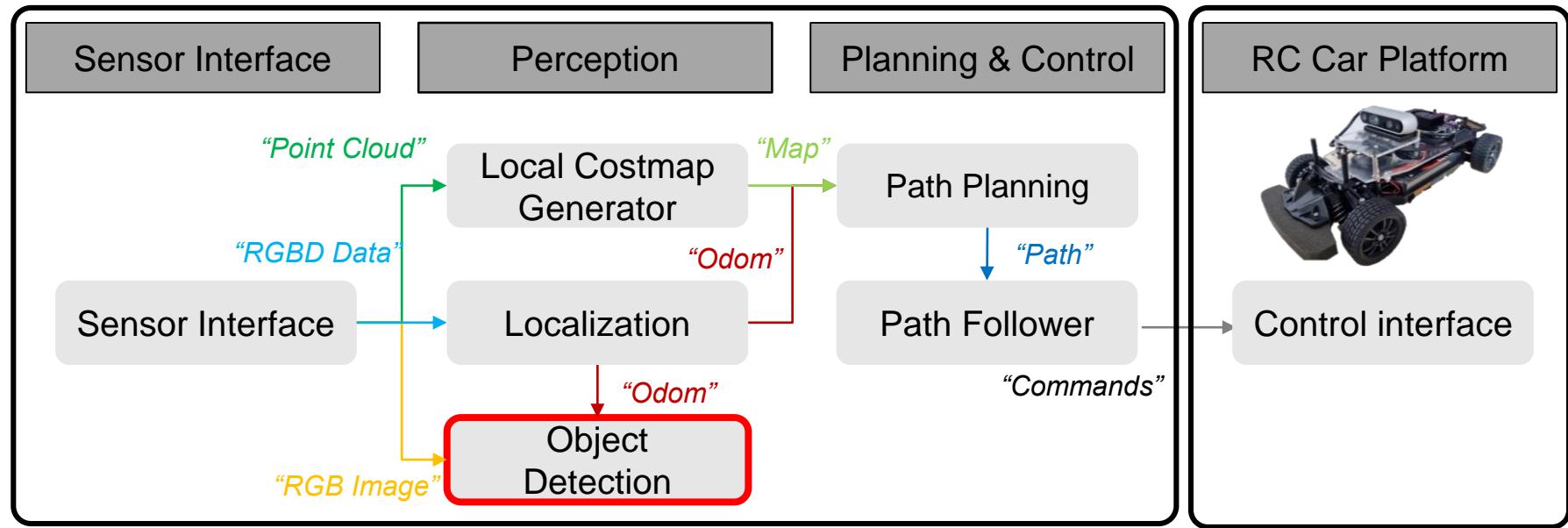


ORB SLAM



Apriltag tracking

# Overall System Architecture



## ➤ Object Detection (darknet\_ros, detected\_image\_saver)

- ❑ Detect object and classify it using YOLO v4 and crop the image using bounding box
- ❑ Estimate the pose of the object using localization result and tag detection
- ❑ Pass (class, position, cropped image) to "detected image saver"

Object detection, Tag detection

Localization

Post processing →

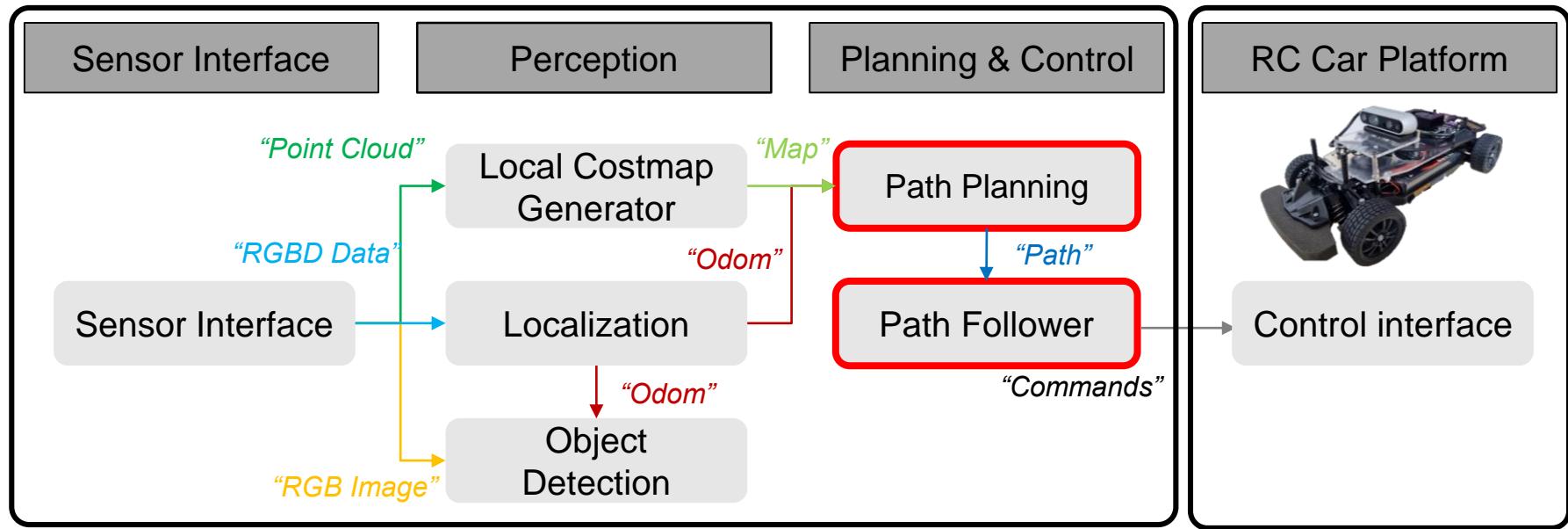
Image saver



darknet\_ros with YOLOv4

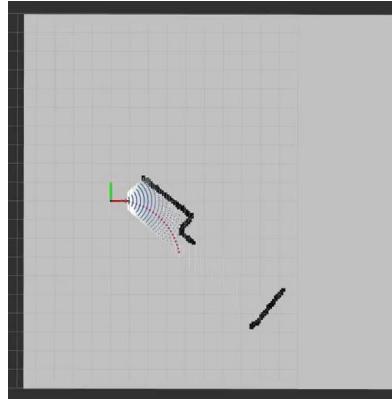
[https://github.com/Tossy0423/yolov4-for-darknet\\_ros](https://github.com/Tossy0423/yolov4-for-darknet_ros)

# Overall System Architecture

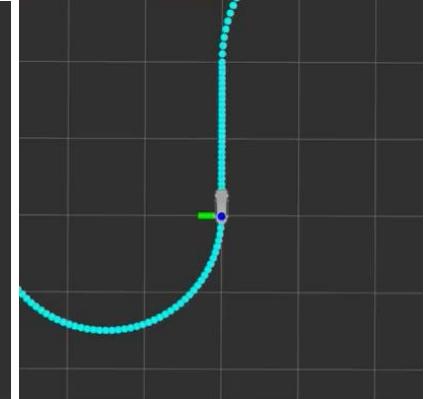


- **Path Planning (`motion_primitives_planner`)**
  - The path planning module generate a **safe path** for a **robot** to navigate environment without collision.
- **Path Follower (`path_follower`)**
  - The path follower module computes **steering angle** and **throttle commands** for the ego vehicle to follow a given path.

[https://github.com/hynkis/motion\\_primitives\\_planner](https://github.com/hynkis/motion_primitives_planner)

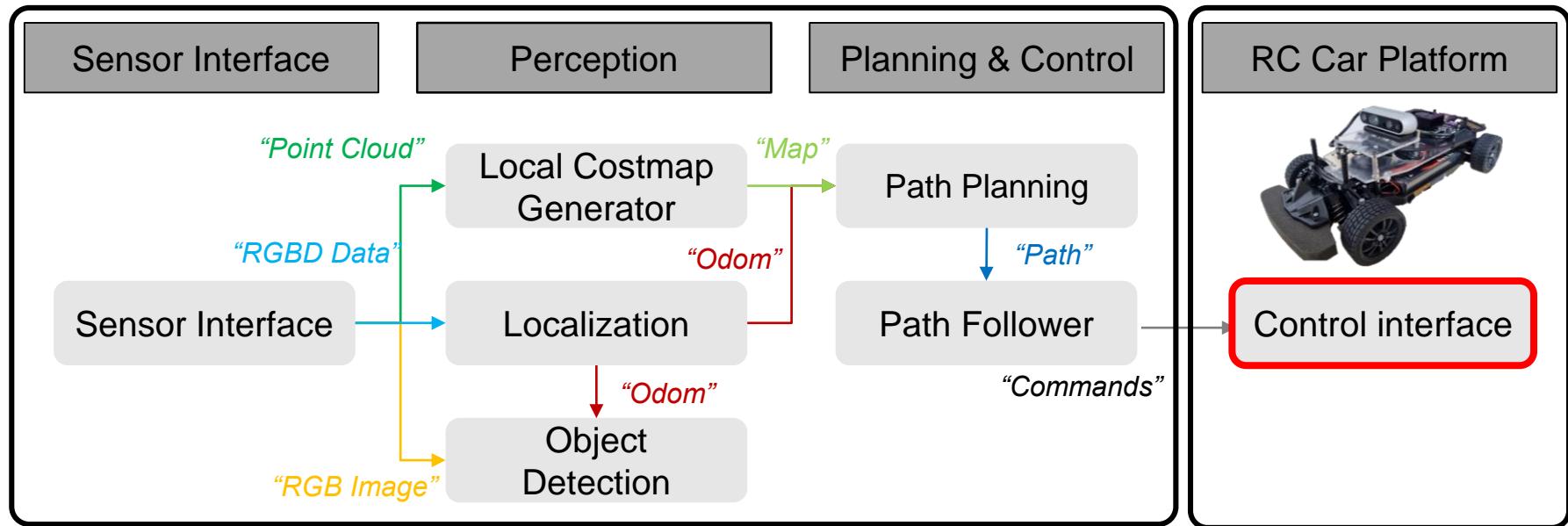


Motion primitives planner



Path following

# Overall System Architecture



## ➤ Control Interface (rosserial\_arduino)

- ❑ The control interface controls **servo motor** for steering and **DC motor** for throttling RC Platform.
- ❑ Use **rosserial\_arduino ROS package** for communicating between ROS system in the Jetson Nx and embedded system in the Arduino board.

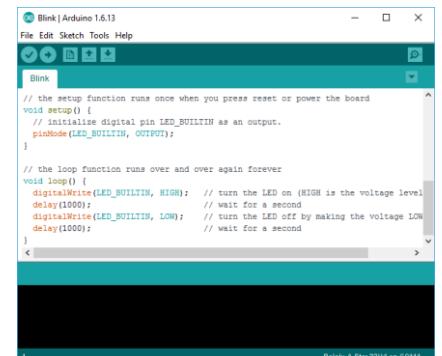


Arduino Nano

**Steering angle**  
(/rc\_cmd/steer)  
(/auto\_cmd/steer)

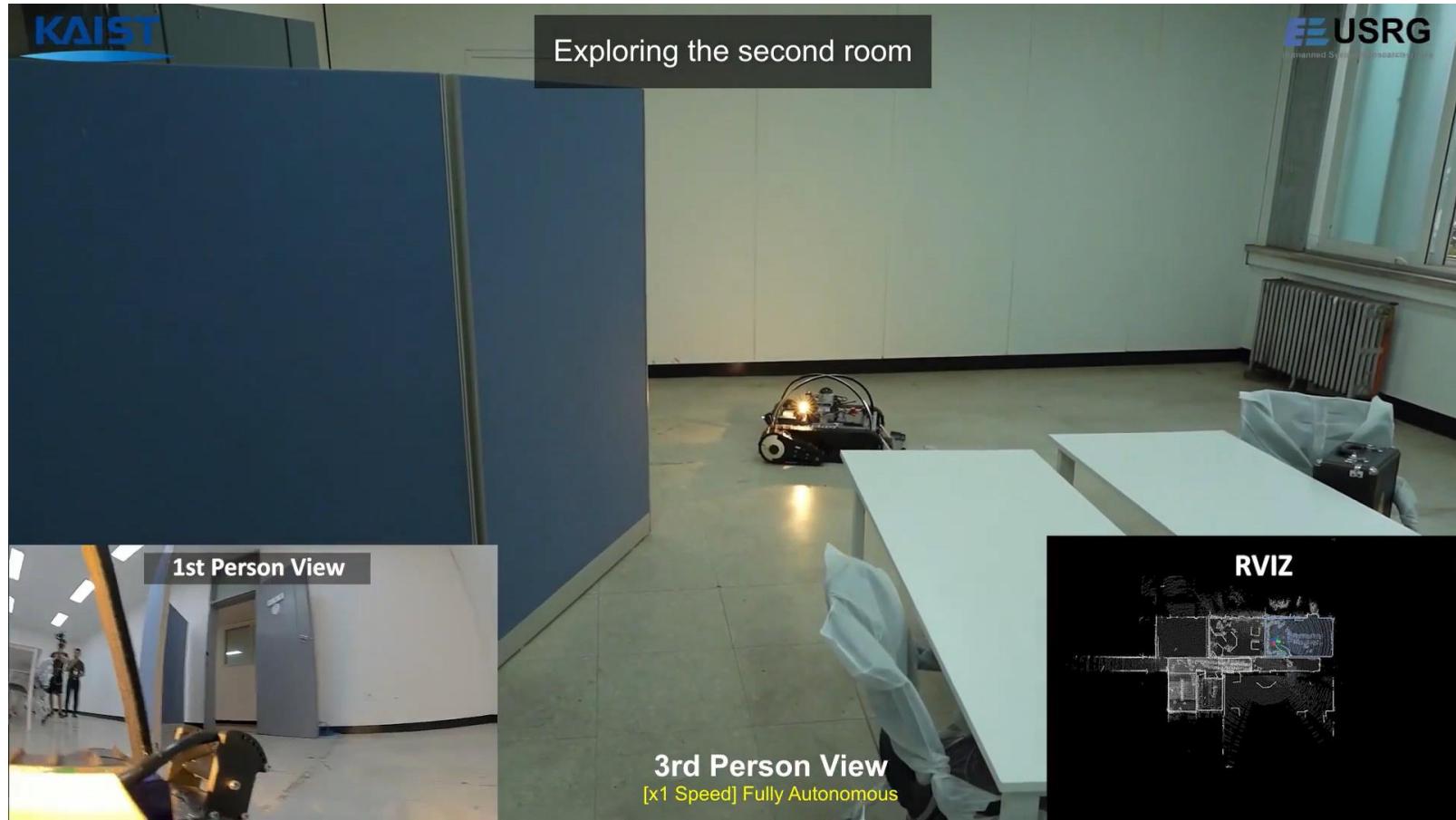
**Throttle**  
(/rc\_cmd/throttle)  
(/auto\_cmd/throttle)

**Driving mode**  
(/auto\_mode)



Arduino IDE

# Exploration Example Video



<https://youtu.be/ua-bf6es4ac?si=2V3C2gu0sI5qZ1E9>

---

---

# Q & A

[hynkis@kaist.ac.kr](mailto:hynkis@kaist.ac.kr)

[fingb20@kaist.ac.kr](mailto:fingb20@kaist.ac.kr)

[sungjae\\_min@kaist.ac.kr](mailto:sungjae_min@kaist.ac.kr)

[hyungjoo\\_kim@kaist.ac.kr](mailto:hyungjoo_kim@kaist.ac.kr)