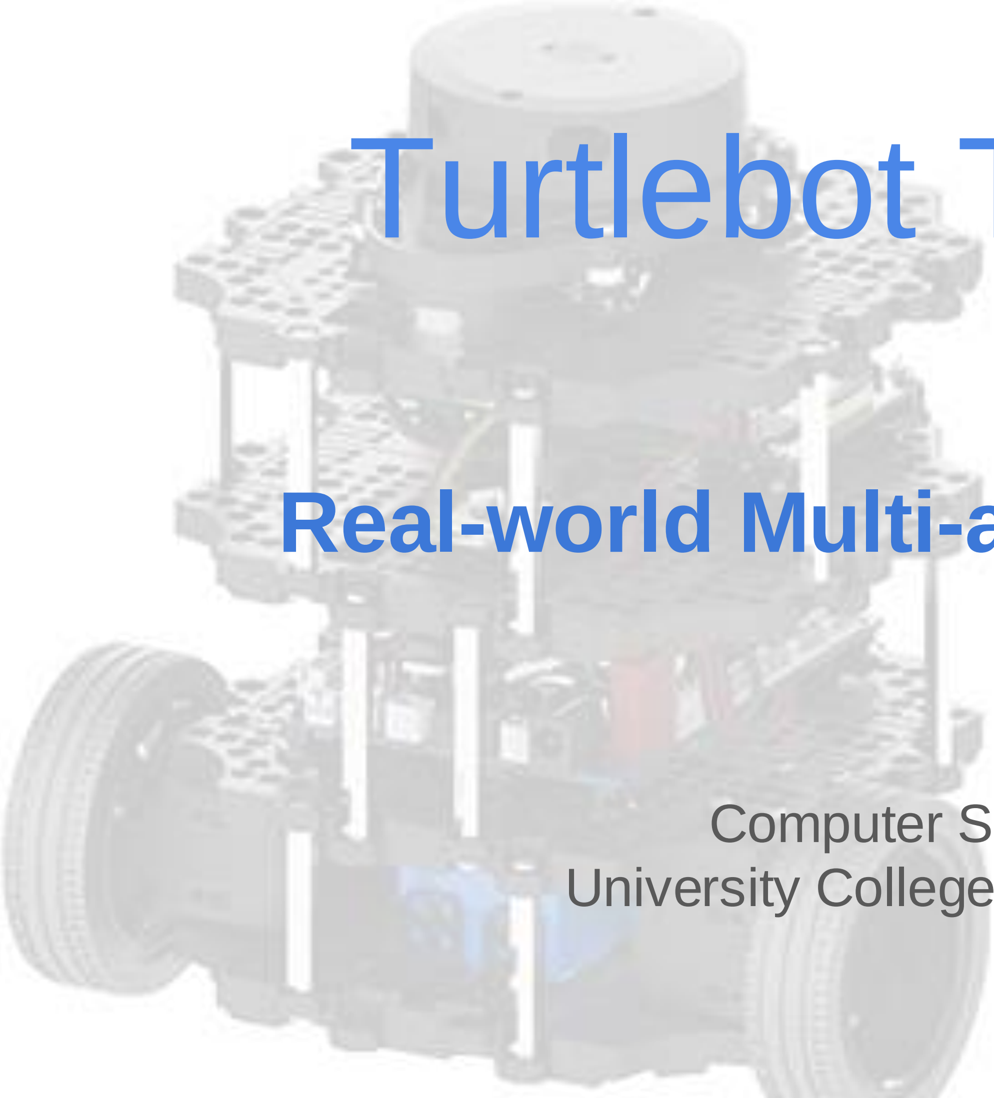
TURTLEBOT3 Burger



Turtlebot Tutorials

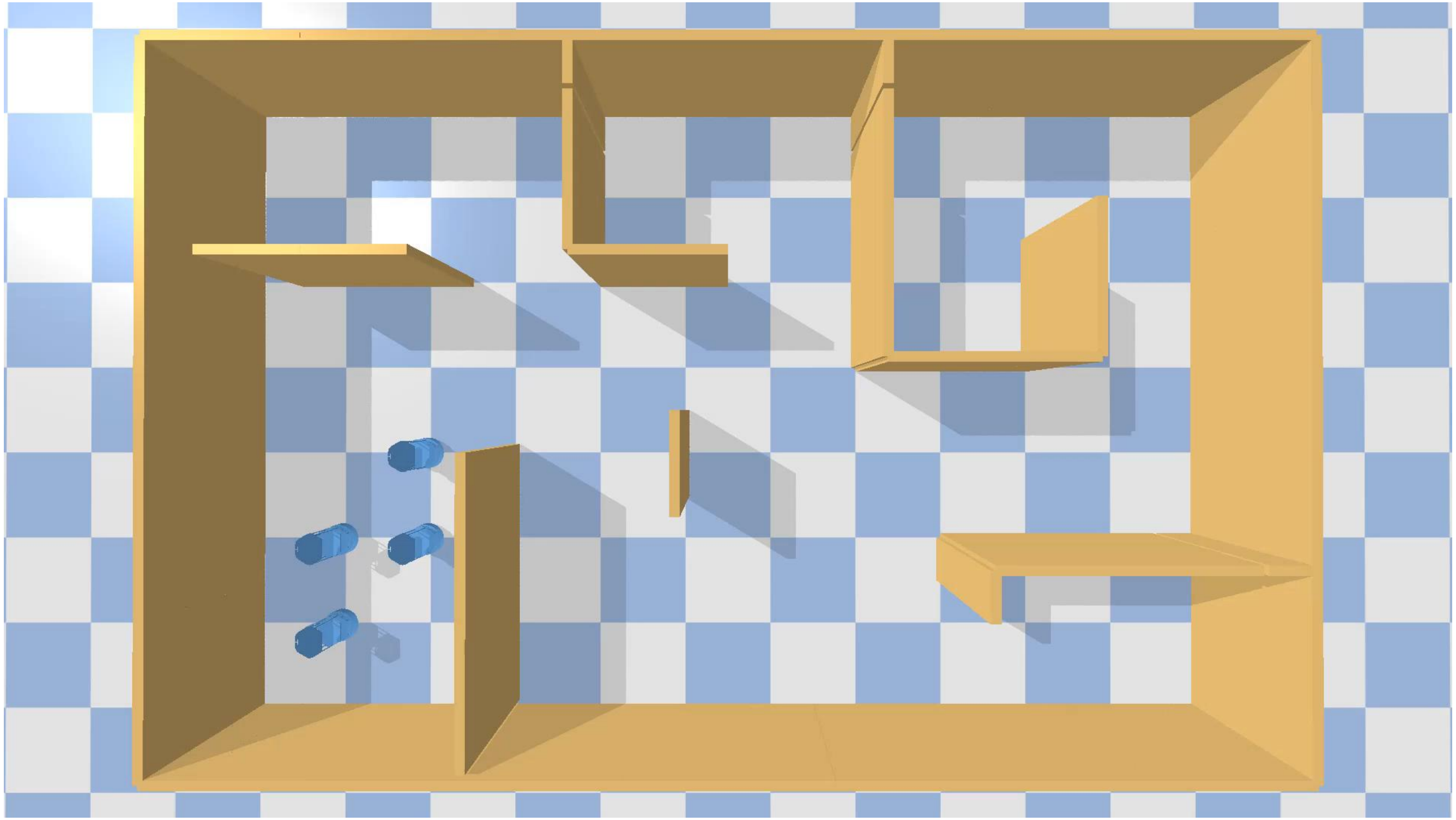
Real-world Multi-agent Systems

Computer Science
University College London, UK

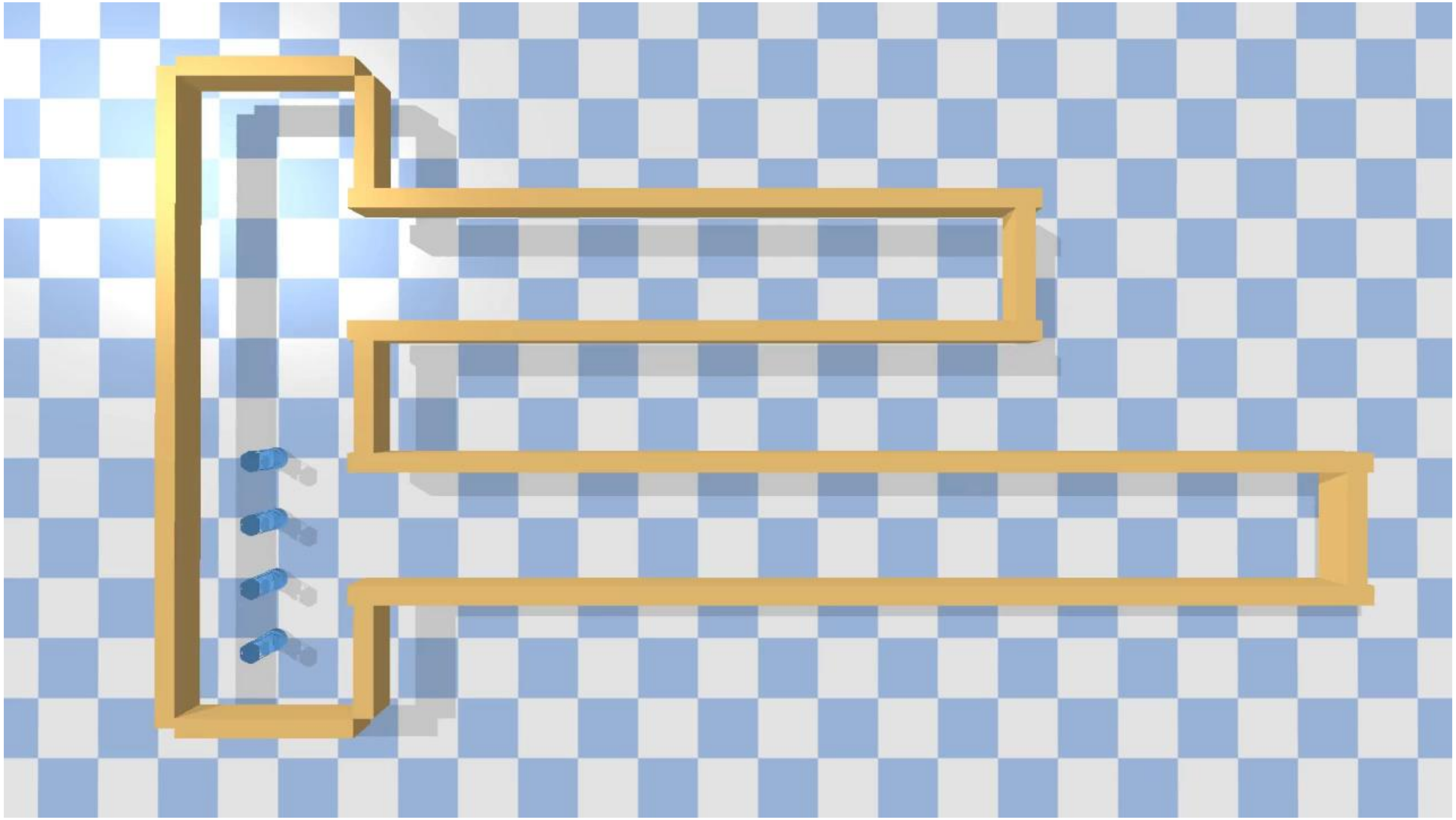
Multi-agent systems

- A multi-agent system (MAS) is a collective of autonomous agents collaborating to achieve shared objectives.
- MAS agents have diverse skills, goals, and decision-making processes.
- Agents can operate in a decentralized or distributed fashion, enhancing adaptability and scalability.

Real-World Multi-Agent Systems



Real-World Multi-Agent Systems



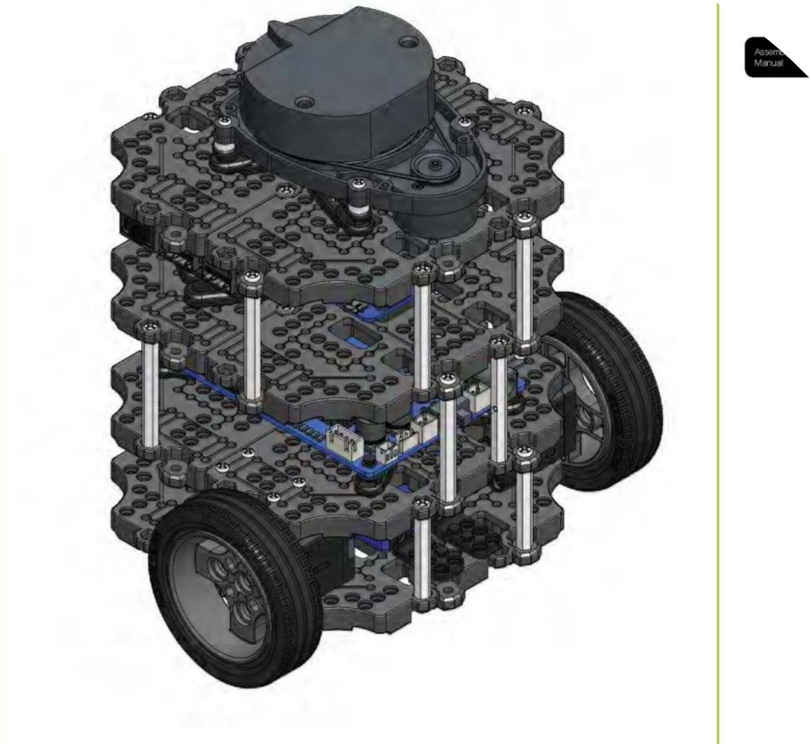
Scope of this tutorial

- How to build a turtlebot? How to perform unit test to make sure it is working
- How to set up pybullet simulation of turtlebots; gitclone and install, set up codebase
- How to test basic motion planning of one turtlebot in simulation, use provided codebase
- How to do one robot’s search and find in simulation , eg a red target block hidden in a room
- How to do multi-robot planning in simulation
- How to do multi-robot search, fetch, return in simulation.
- How to do motion planning of one **real robot**
- How to do motion planning of multiple real **robots**

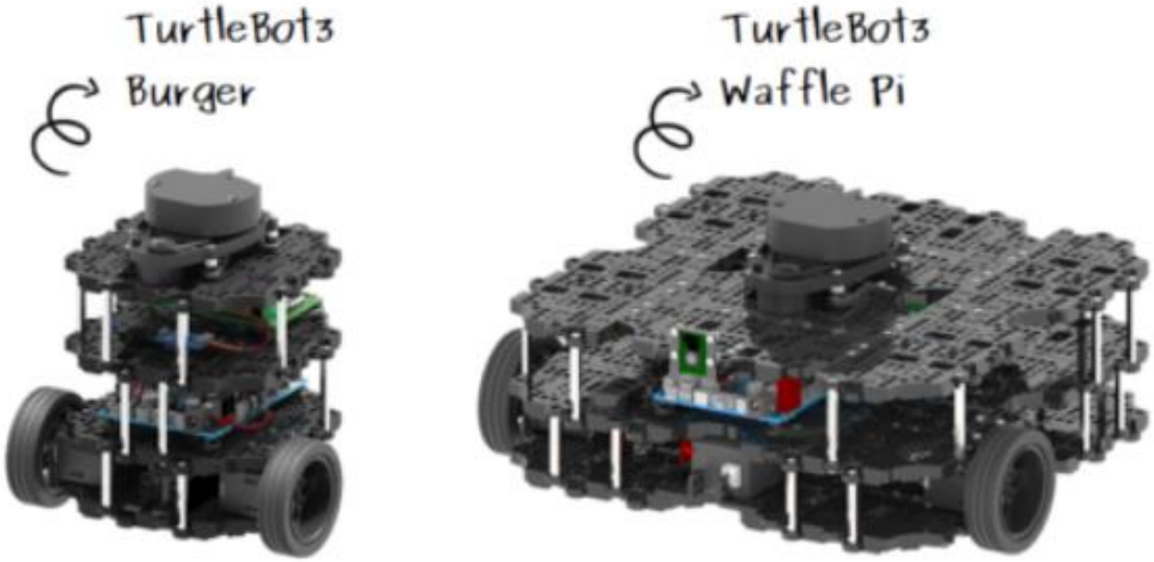
Turtlebot3 burger assembly

- Main landing page: https://emanual.robotis.com/docs/en/platform/turtlebot3/hardware_setup/

- Assembly manual to download PDF: http://www.robotis.com/service/download.php?no=748



3. 4. Hardware Assembly



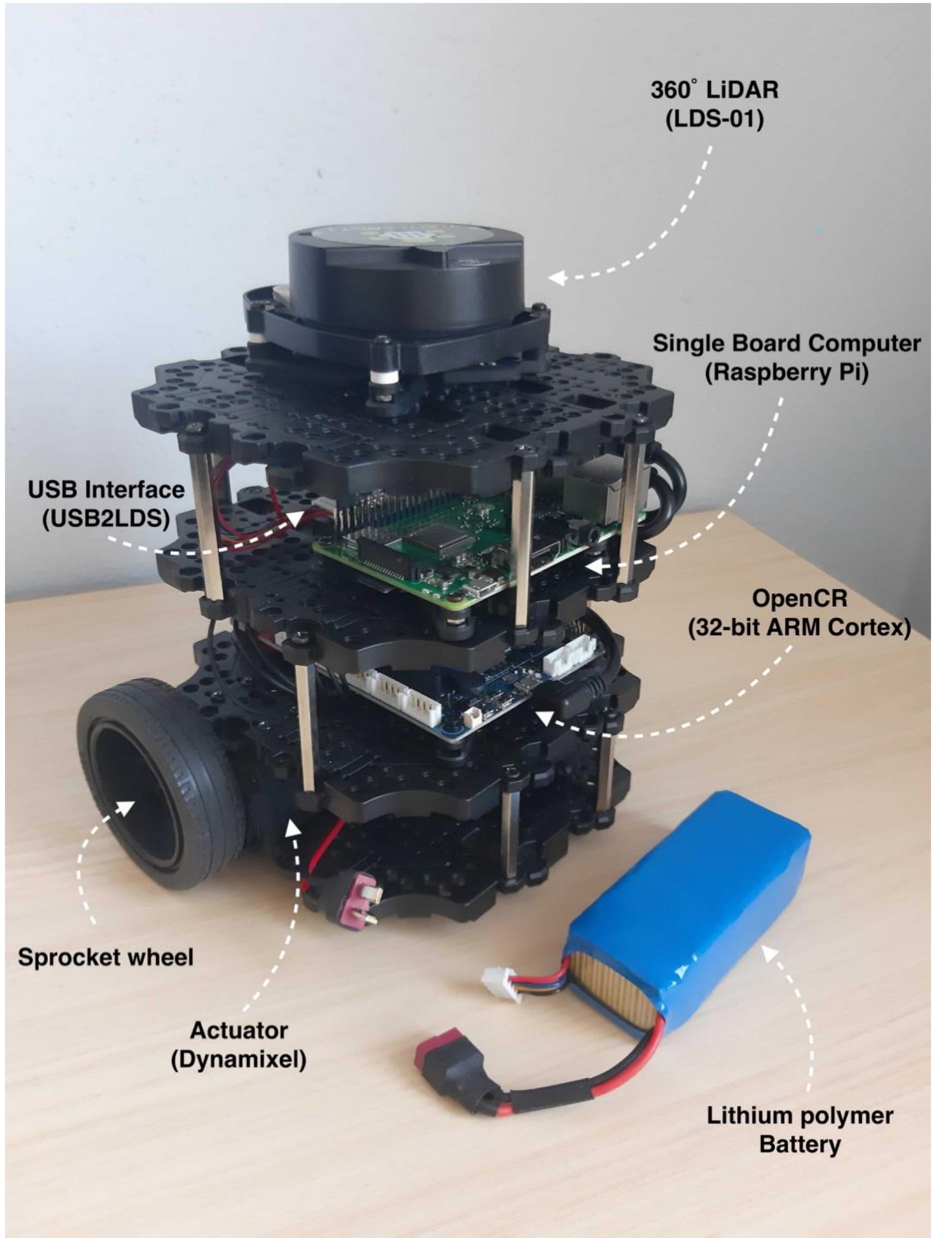
3. 4. 1. Assembly Manual

TurtleBots3 is delivered as unassembled parts in the boxes. Follow the instructions to assemble TurtleBot3.

- Download PDF Assembly manual for TurtleBot3 Burger
- Download PDF Assembly manual for TurtleBot3 Waffle
- Download PDF Assembly manual for TurtleBot3 Waffle Pi

TurtleBot3: A ROS Standard Platform Robot

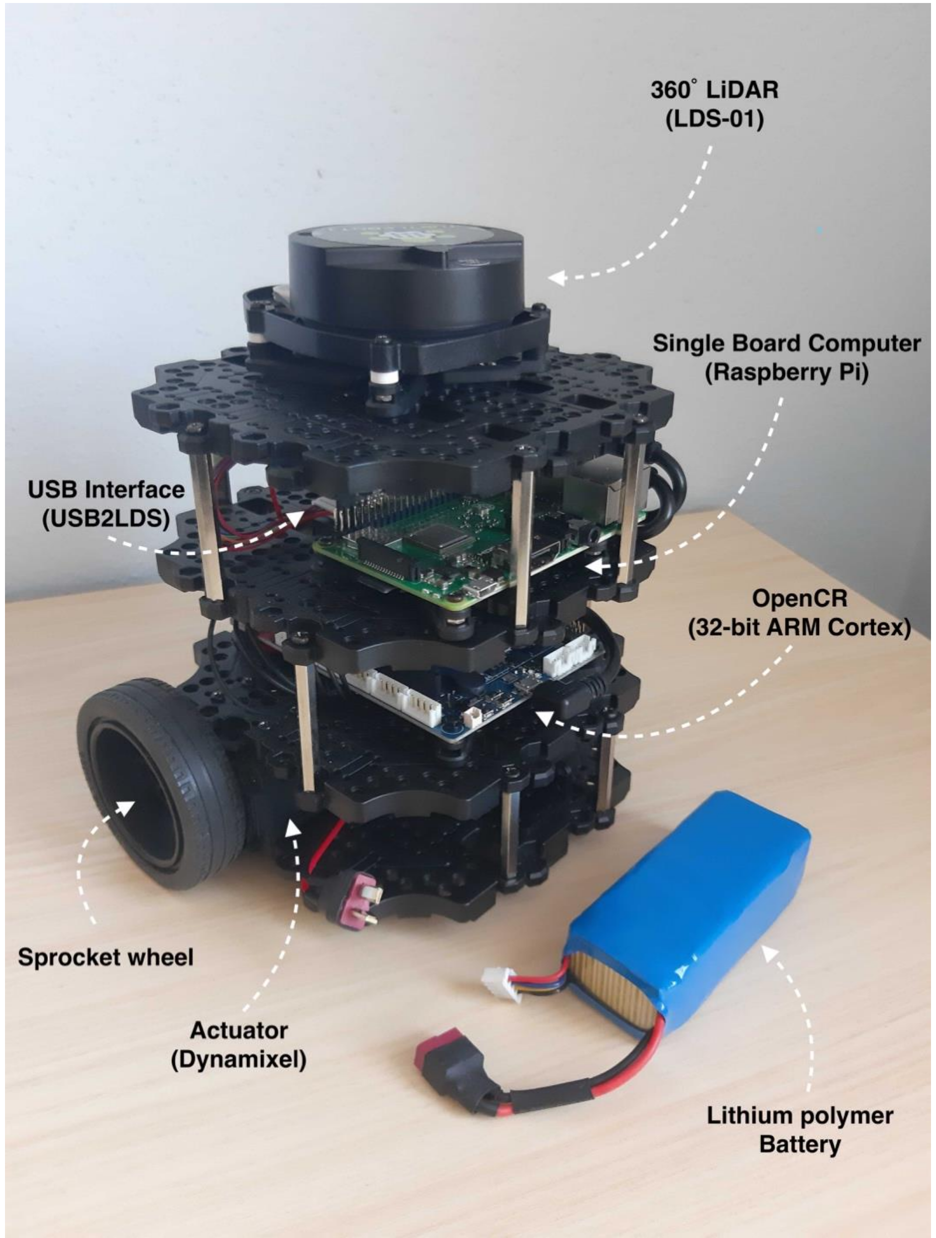
- Easy Assembly
- Programmable, Extensible Architecture
- Cost-Effective
- Detailed Documentation
- 2-5 Hours of Assembly



TurtleBot3: A ROS Standard Platform Robot

Four-Layered Design, bottom up

- Layer 1: Robot Actuators, Battery, and Wheels
- Layer 2: Open-Source ROS Control Module
- Layer 3: Raspberry Pi 3 and USB Interface
- Layer 4: 360-degree LiDAR



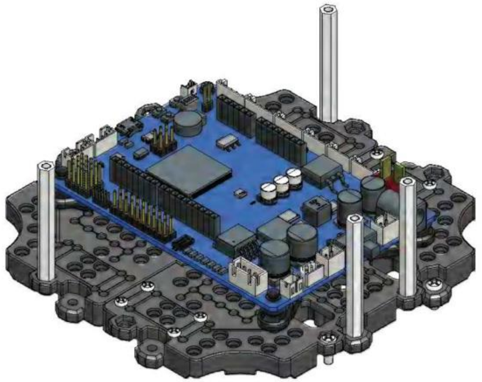
Layer 1

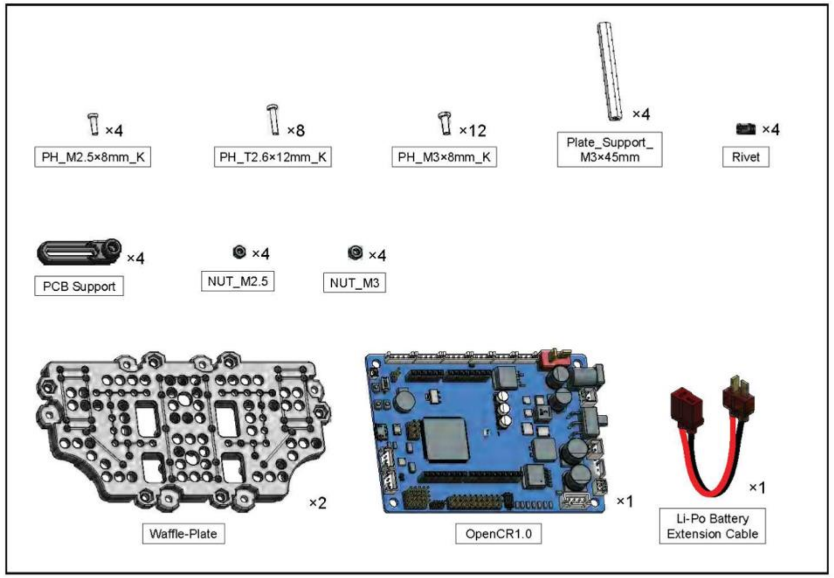

TURTLEBOT3
First Layer Assembly
English
中文
日本語
한국어
Assembly Manual
×8
PH_M2×4mm_K
×4
PH_T2×6mm_K
×8
PH_T2.6×12mm_K
×8
PH_M3×8mm_K
×4
NUT_M3
×4
Plate_Support_M3×35mm
×2
DYNAMIXEL (XL430)
×2
DYNAMIXEL to OpenCR Cable
×1
Ball Caster
×2
Wheel
×2
Tire
×2
Waffle-Plate
×1
Li-Po Battery
×10
Rivet
×5
Bracket
11

Layer 2

Burger

Second Layer Assembly

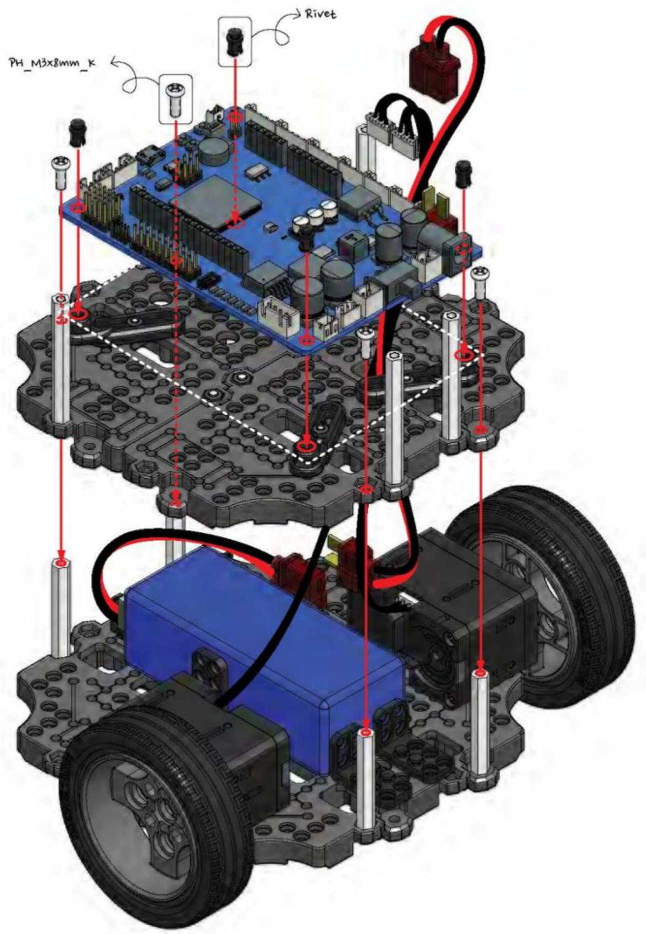




Burger

Mount OpenCR1.0 on PCB Supports

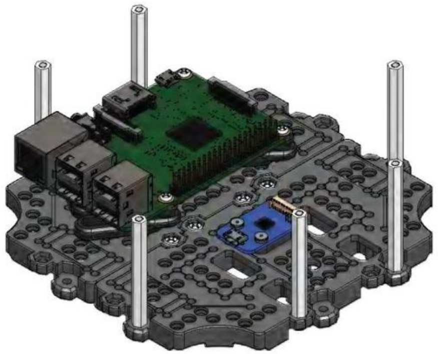
Assemble the Second Layer on top of the First Layer

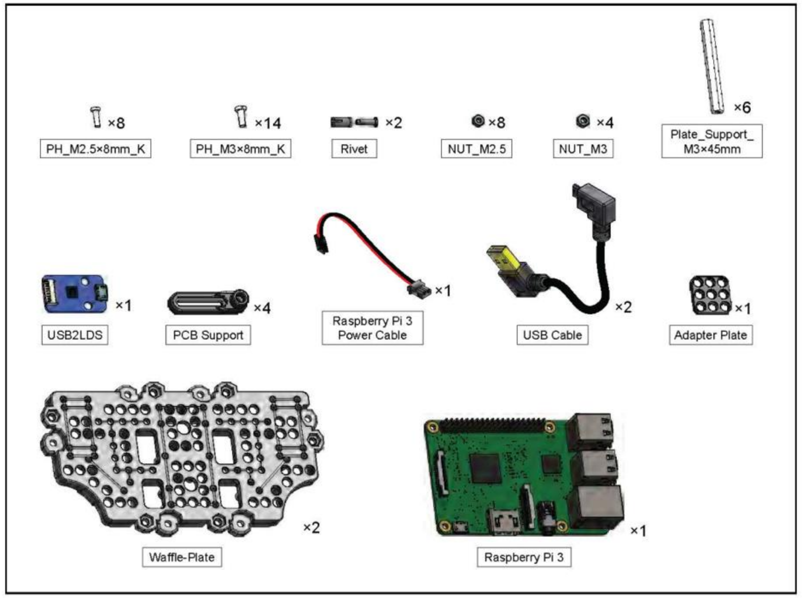


Layer 3

Burger

Third Layer Assembly



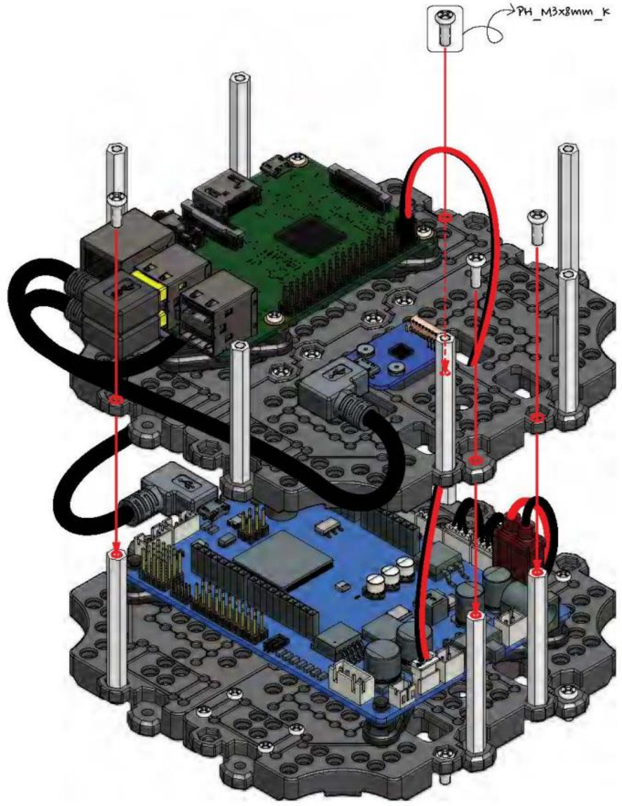


20

TURTLEBOT3

English
中文
日本語
한국어
Assembly Manual

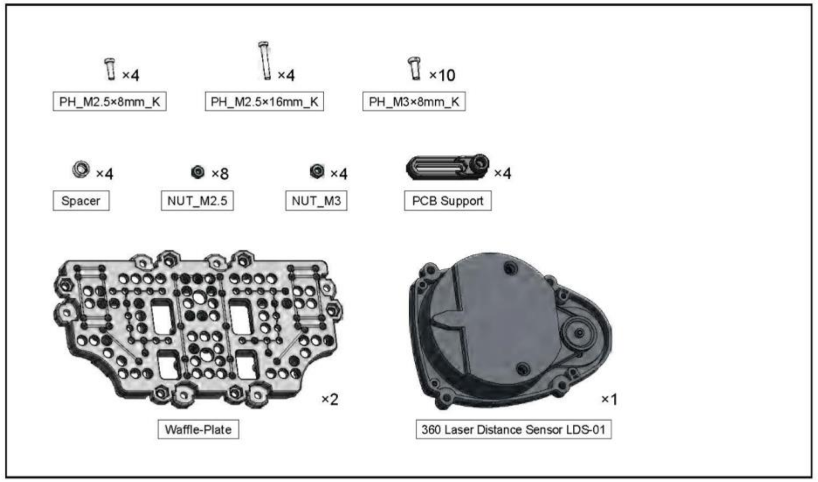
Assemble the Third Layer on top of the Second Layer



Layer 4

Burger

Fourth Layer Assembly



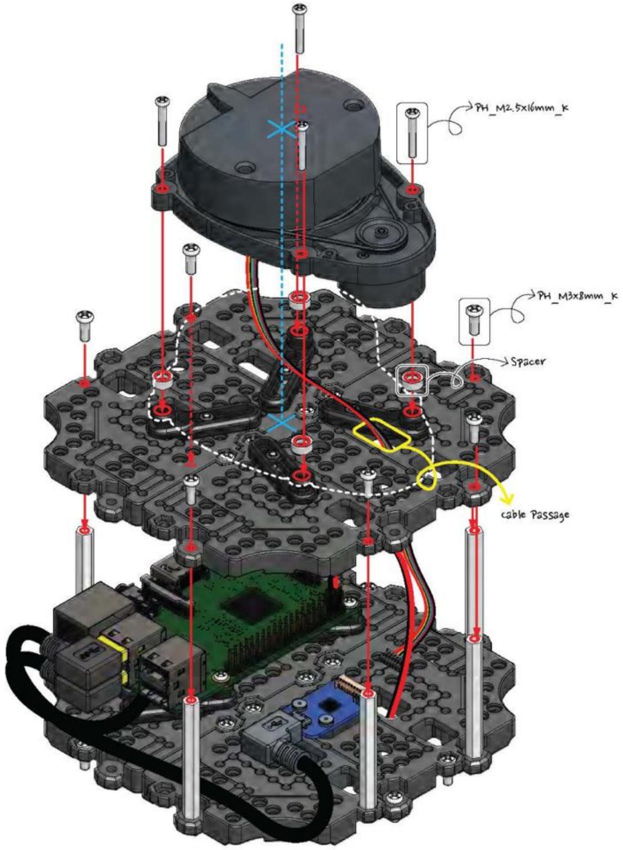
Burger

Place Spacers on PCB Support

Assemble 360 Laser Distance Sensor LDS-01 on top of Spacers

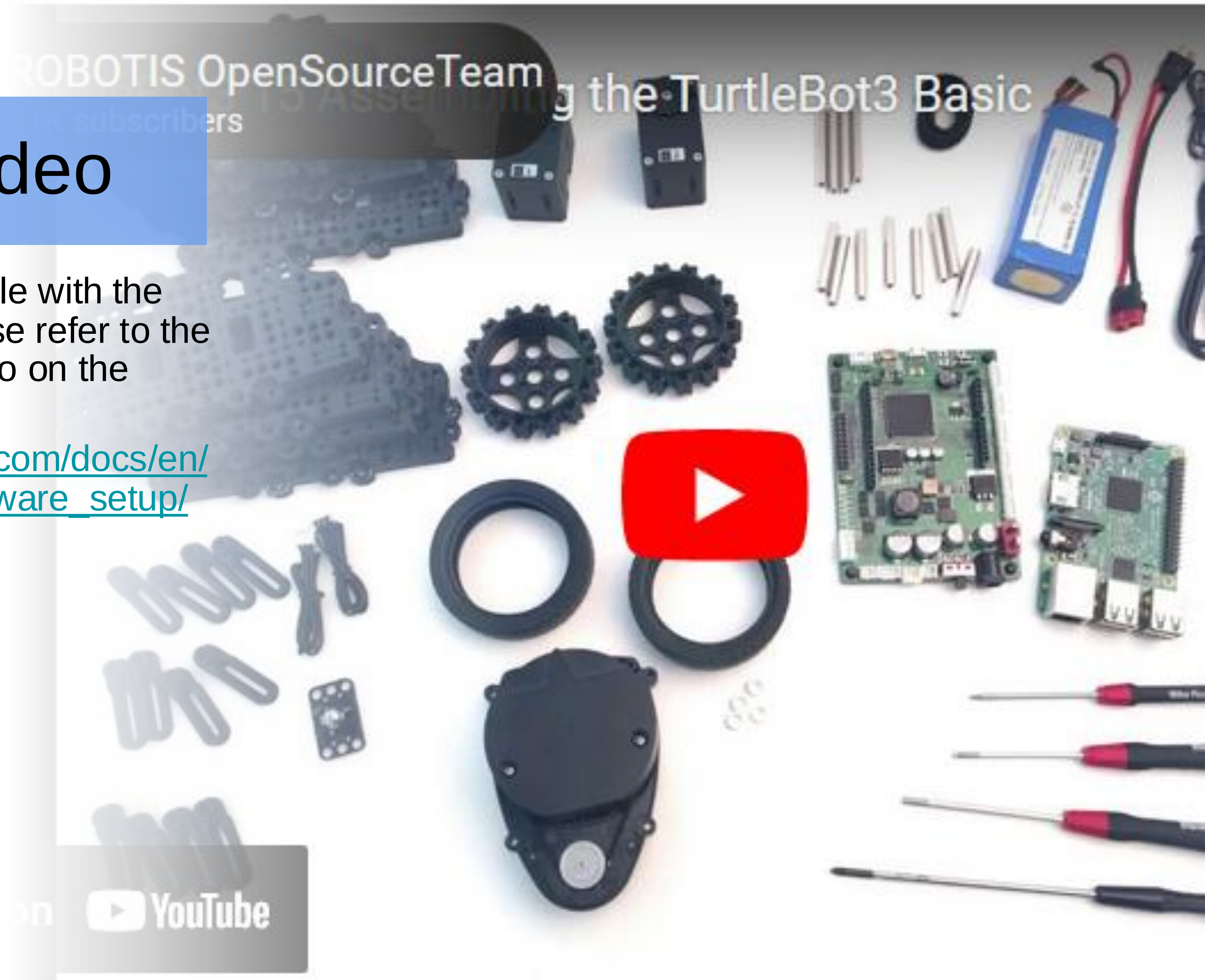
Assemble Fourth Layer on top of the Third Layer

Connect 360 Laser Distance Sensor LDS-01 to USB2LDS on the Third Layer



Assembly Video

- If it is difficult to assemble with the assembly manual, please refer to the following assembly video on the website:
- https://emanual.robotis.com/docs/en/platform/turtlebot3/hardware_setup/



SBC and OpenCR setup

- Official website: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

- Manual: https://github.com/JacksonChiy/turtlebot_manual

Pybullet simulation setup

- Repository setup:

```
git clone "link"
```

- Environment setup:

```
cd "Directory name"
python3 -m pip install -r
requirement.txt
```

Original link: https://github.com/JacksonChiy/turtlebot_simulation_pybullet

UCL

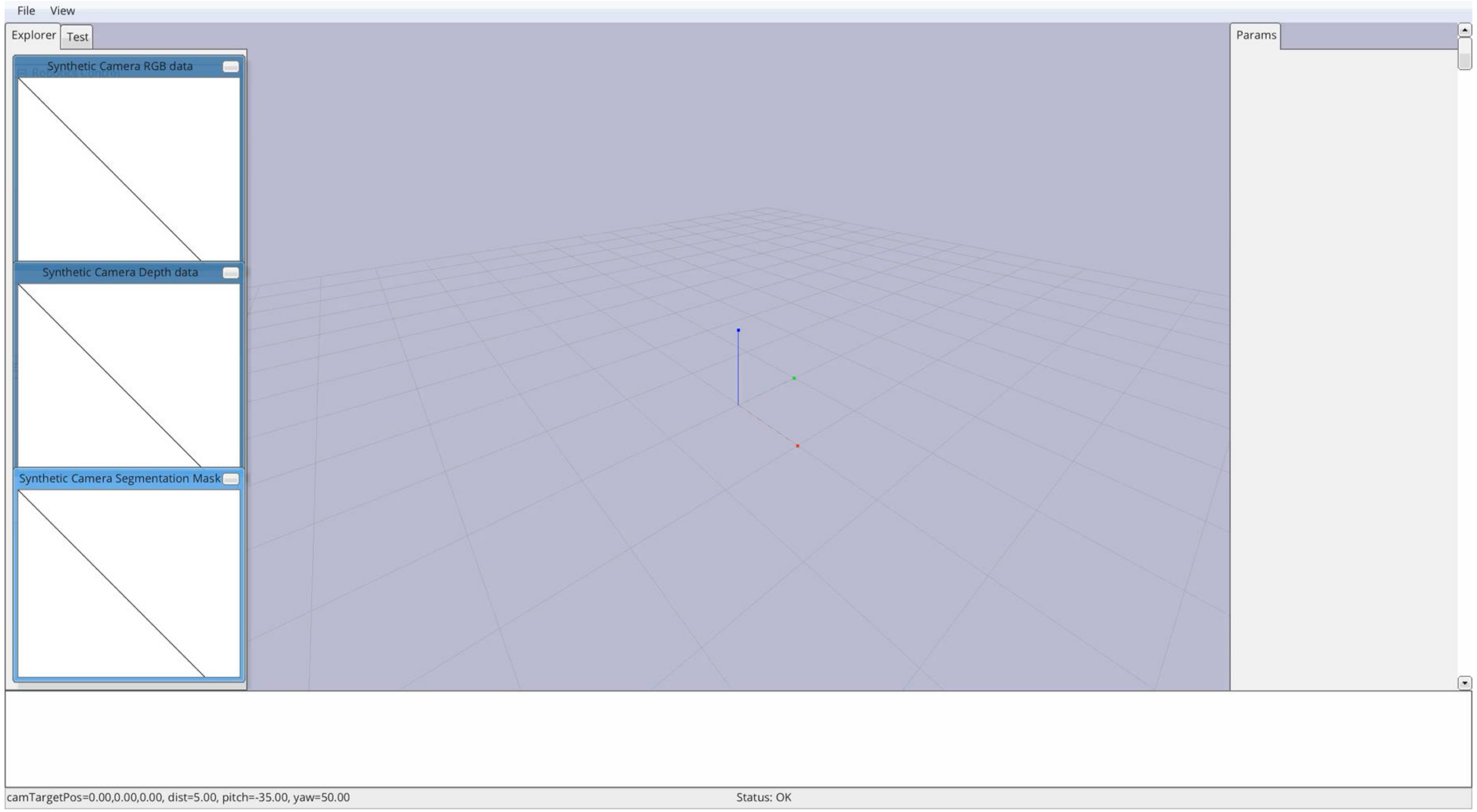
Motion control testing for single turtlebot

- Run motion control test

```
cd “Directory name”
python3 single_bot_motion_control.py
```

- Change goal point

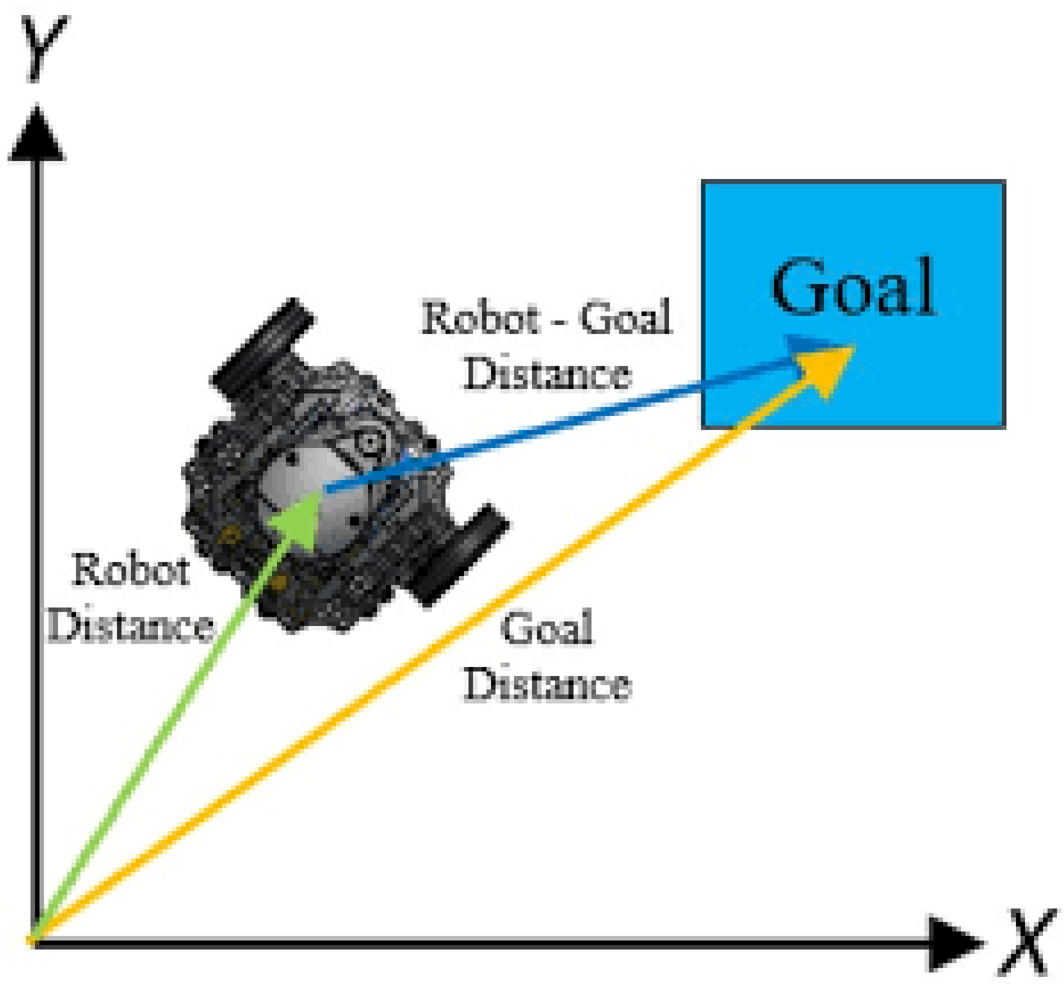
```
def goto(agent, goal_x, goal_y):
goto(boxId, goal_x: 1, goal_y: 1)
```



Motion control for one turtlebot

Control actions/effort: driving speed of wheels

- Decoupled into ***linear driving velocity*** and ***angular turning velocity***



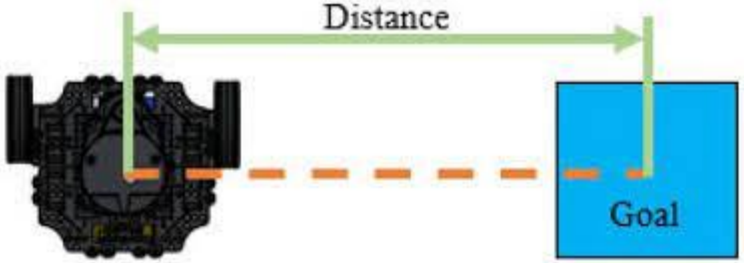
Motion control for one turtlebot

Control actions/effort: driving speed of wheels

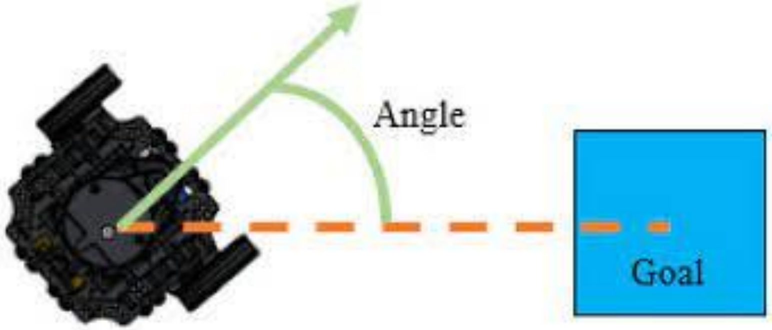
- Decoupled into ***linear driving velocity*** and ***angular turning velocity***

Control rules

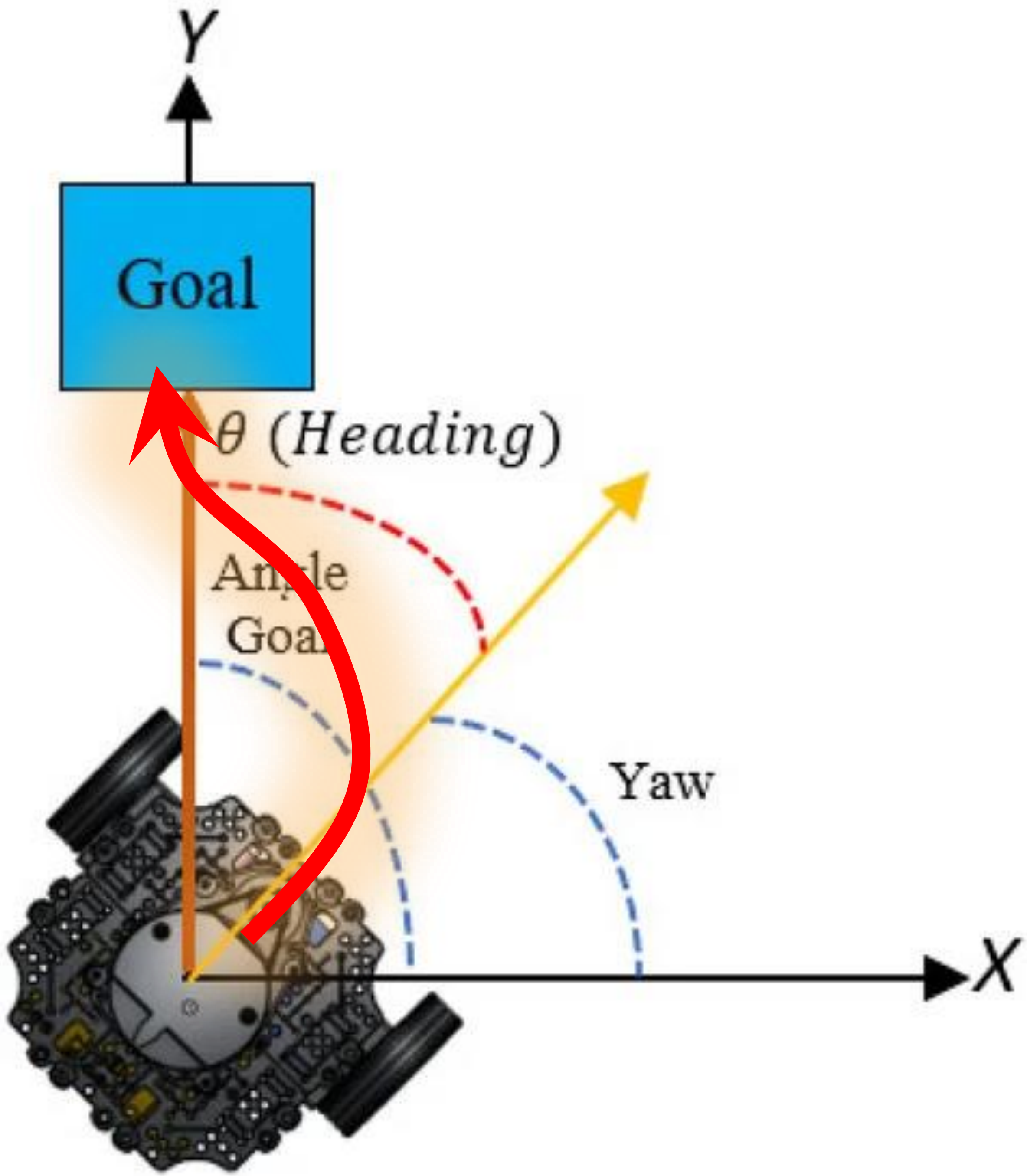
- Linear driving velocity is proportional to the ***distance error***, and capped at a maximum velocity limit.



- Turning/Rotational speed is proportional to ***angular error***, and capped at a maximum angular rate.

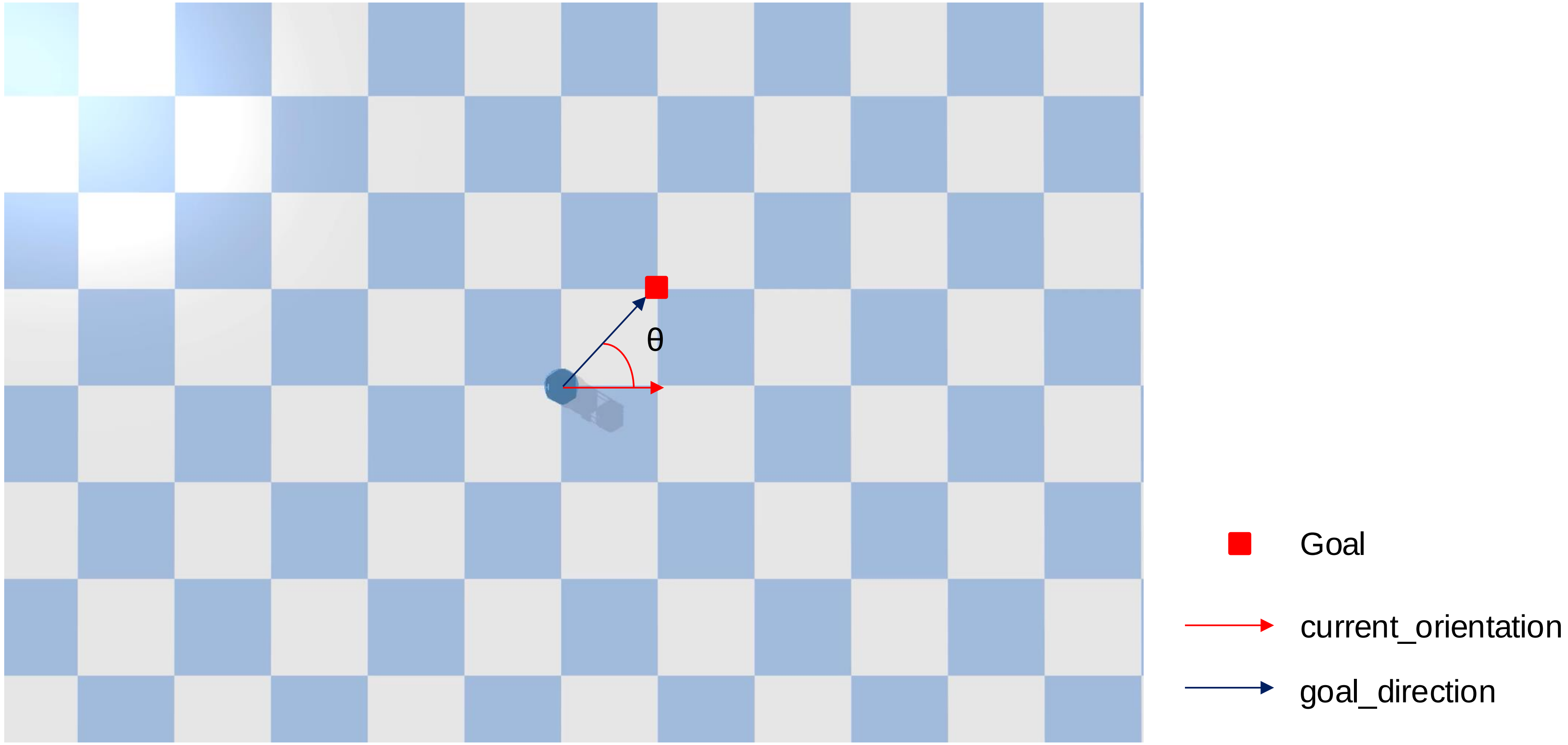


Motion control for one turtlebot



Theta

- How to calculate theta
  - goal_direction – current_orientation



Motion control for one turtlebot

- In **single_bot_motion_control.py**
- Angular velocity: Kp_angular * theta
- Linear velocity: Kp_linear * math.cos(theta) → **max(0, Kp_linear * math.cos(theta))**

```
k_linear = 10
k_angular = 30
linear = k_linear * math.cos(theta)
angular = k_angular * theta

rightWheelVelocity = linear + angular
leftWheelVelocity = linear - angular
```

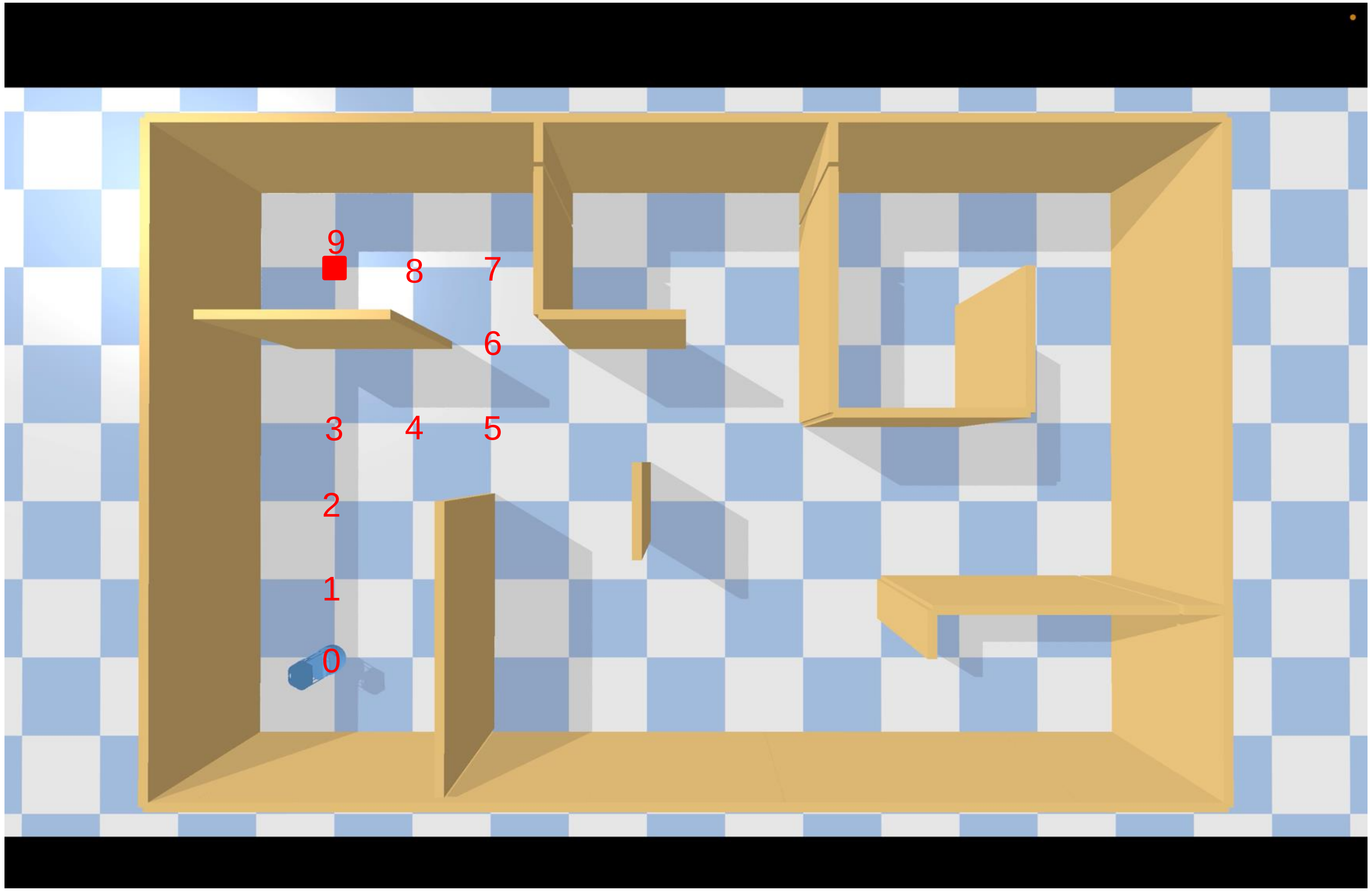
Single turtlebot search and find in simulation

- Task 1: Target Search
  - *Objective:* Each bot will embark on a mission to search a path for hidden targets within their assigned rooms, utilizing advanced path finding algorithms like Dijkstra or A*.

- Task 2: Fetch and Return
  - *Objective:* After successfully locating their respective targets, the bots embark on a mission to fetch the targets and return them to the base using motion control based on the path found in task 1.

Single turtlebot search and find in simulation

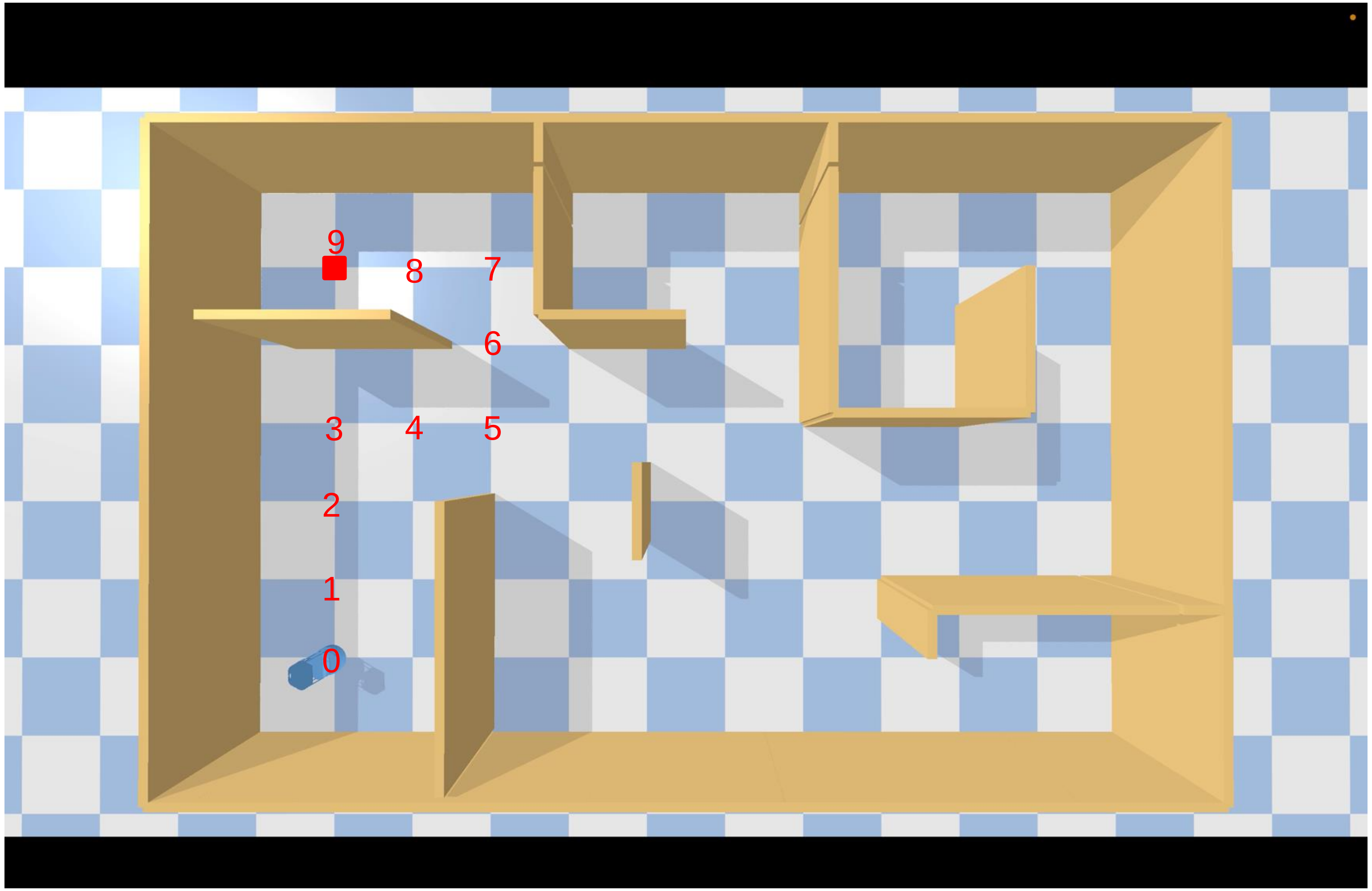
- Path finding



Goal

Single turtlebot search and find in simulation

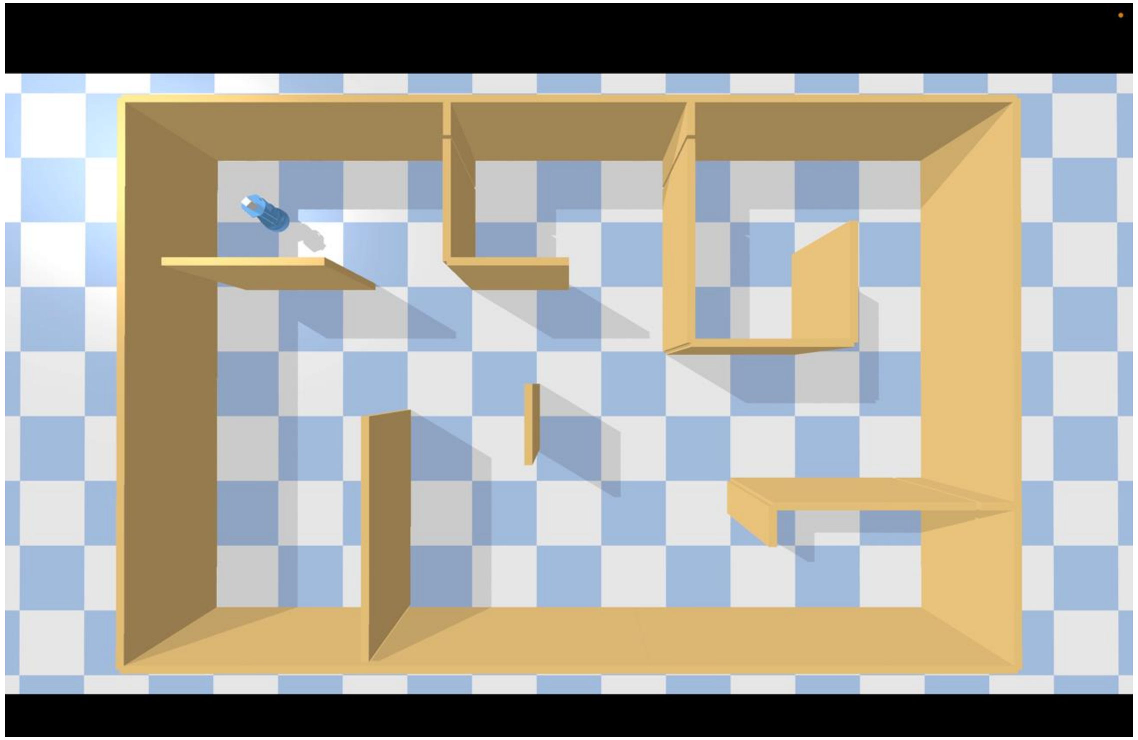
- Motion control: let the turtlebot follow the list of via points from 0 to 9 to fetch target



Goal

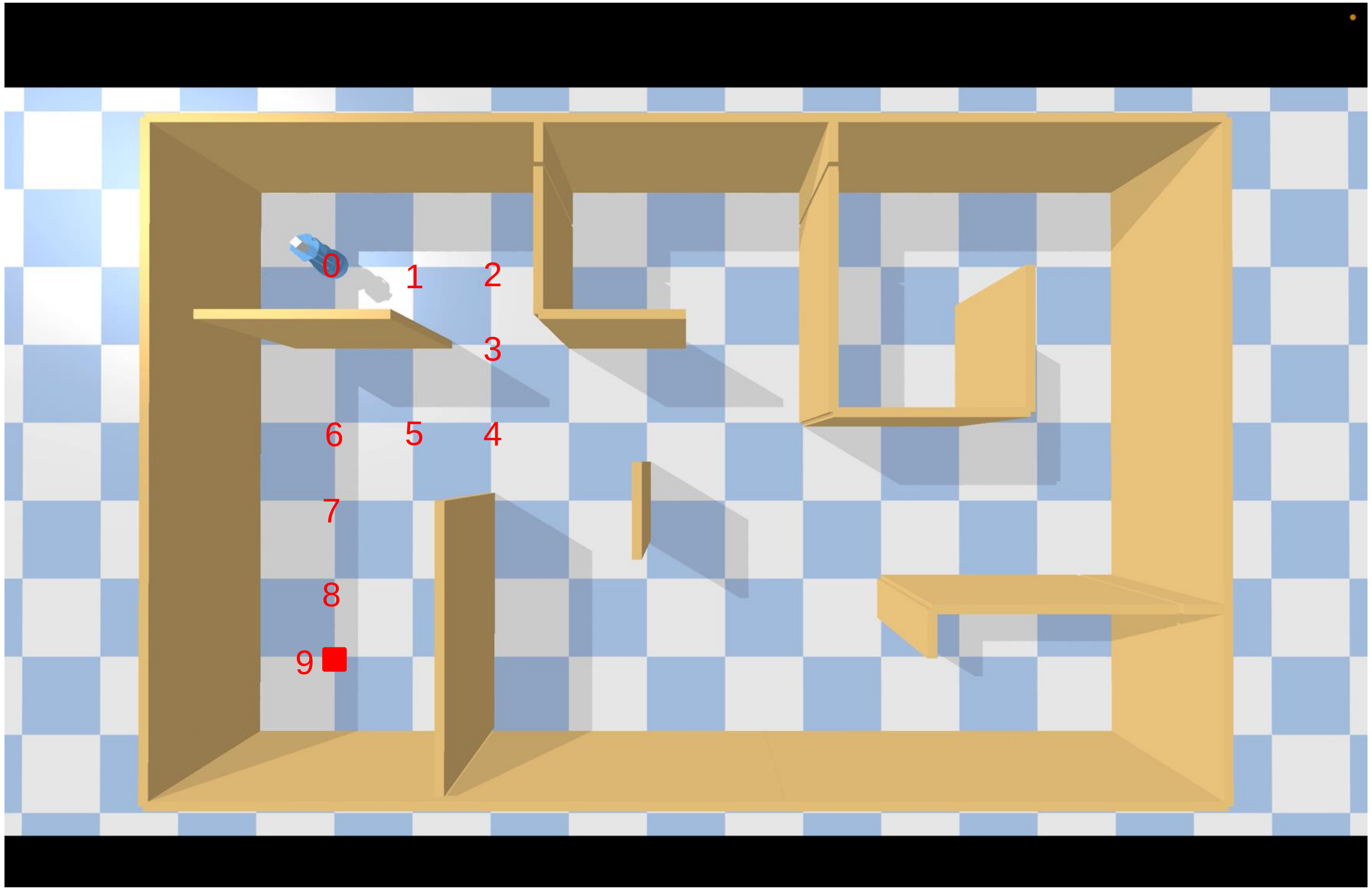
Single turtlebot search and find in simulation

- Fetch target:
  - ❑ Upon reaching their goal positions, the robots employ a simplified approach for target retrieval in the simulation.
  - ❑ Targets are represented by dropping them directly onto the robots, symbolizing the successful acquisition.



Single turtlebot search and find in simulation

- Return: reverse the order of via points for robots to return to the start position



Start position

Multi-robot planning in simulation

Multi-robot planning in simulation

Example algorithm:

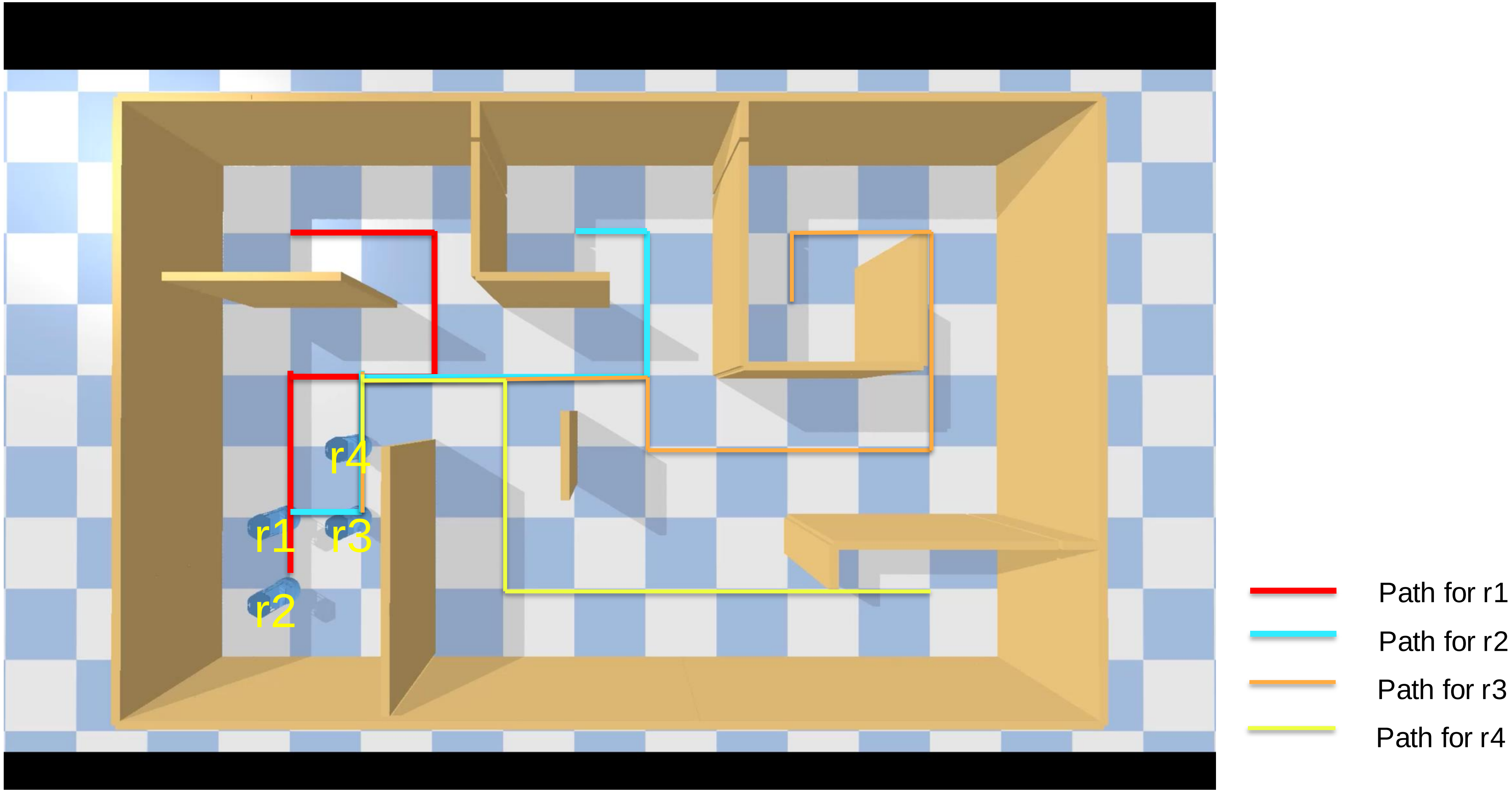
- *Conflict-based search(CBS) for optimal multi-agent pathfinding*

2 Levels for CBS:

- At the high level, a search is performed on a *Conflict Tree* (CT) which is a tree based on conflicts between individual agents. Each node in the CT represents a set of constraints on the motion of the agents.
- At the low level, fast single-agent searches are performed to satisfy the constraints imposed by the high-level CT node. In many cases, this two-level formulation enables CBS to examine fewer states than A* while still maintaining optimality. We analyze CBS and show its benefits and drawbacks.

Reference https://www.sciencedirect.com/science/article/pii/S0004370214001386

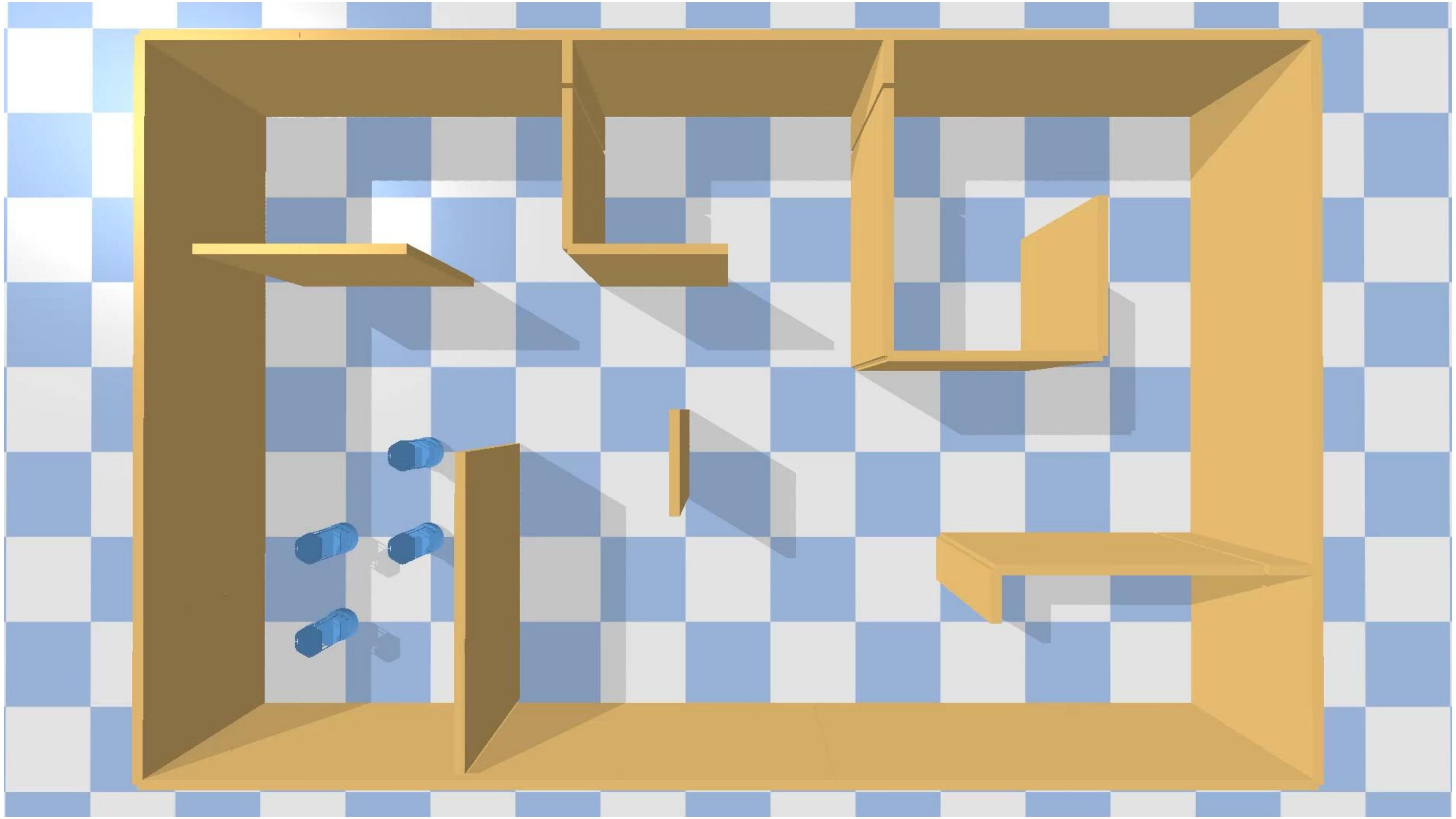
Results of CBS



Multi-robot search, fetch and return in simulation

- Each robot has its own list of via points generated by CBS.
- Employ the same motion control for each robot. (Multi-threading can be used.)
- Employ the same target fetching mechanism for each robot.
- Return: Apply CBS again to generate path for every robot from their goal positions to their start positions.

Multi-robot search, fetch and return in simulation



Single-/Multi-robot control & planning on real robots

Motion planning for single real robot

- Overall setting:

Camera

Turtlebot3 burger

ArUco marker

Motion planning for single real robot

- Motion control algorithm: same as that for simulation for single robot.

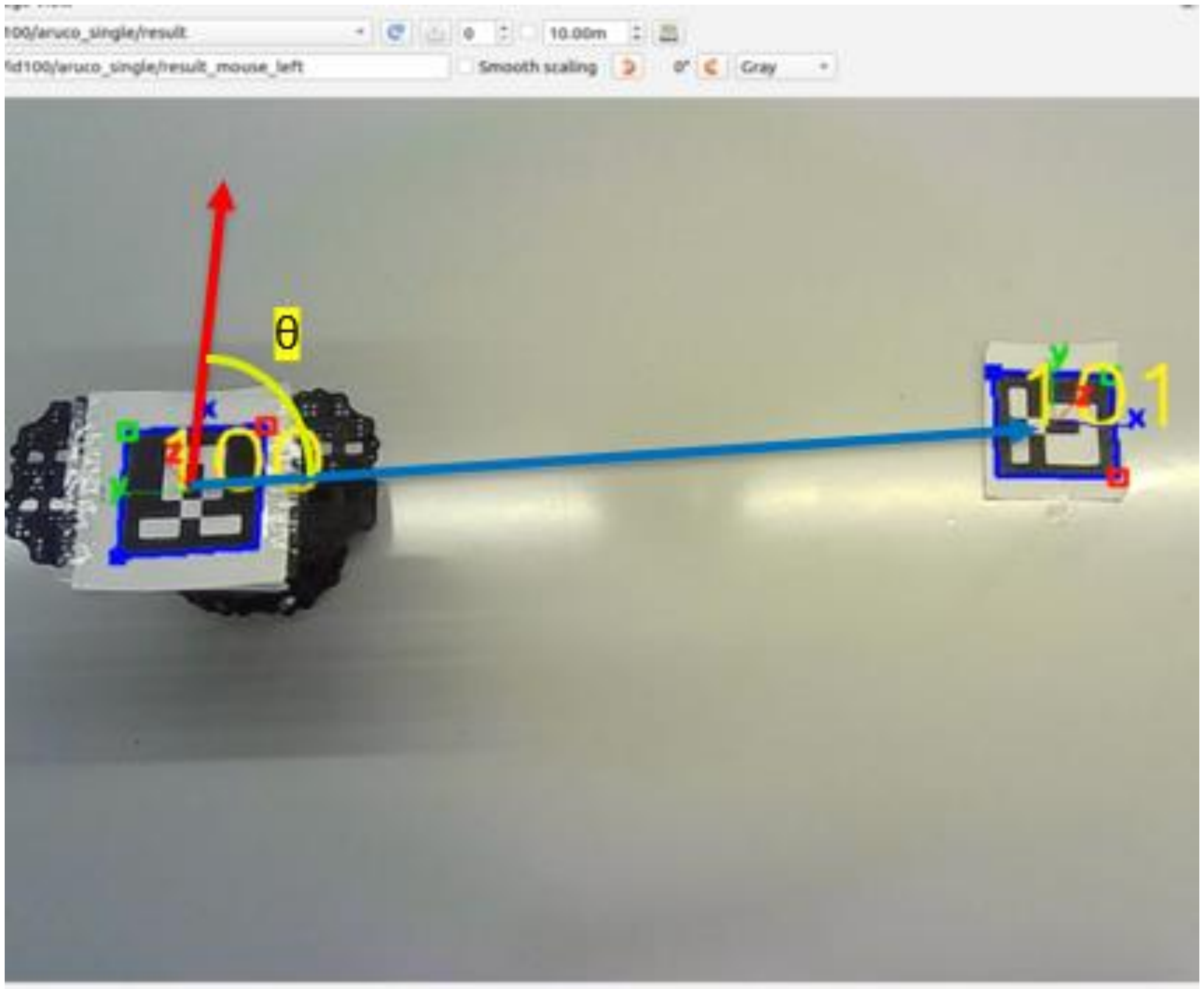
- ArUco marker needs to be stuck on the turtlebot correctly so that the orientation of the ArUco marker is the same as the orientation of the turtlebot. The orientation of the marker can be read by the camera.

- Topics are used in ROS for communications between the turtlebot and the master PC. For example, the topic ‘cmd_vel’ is used to control the velocity of the wheels for the turtlebot.

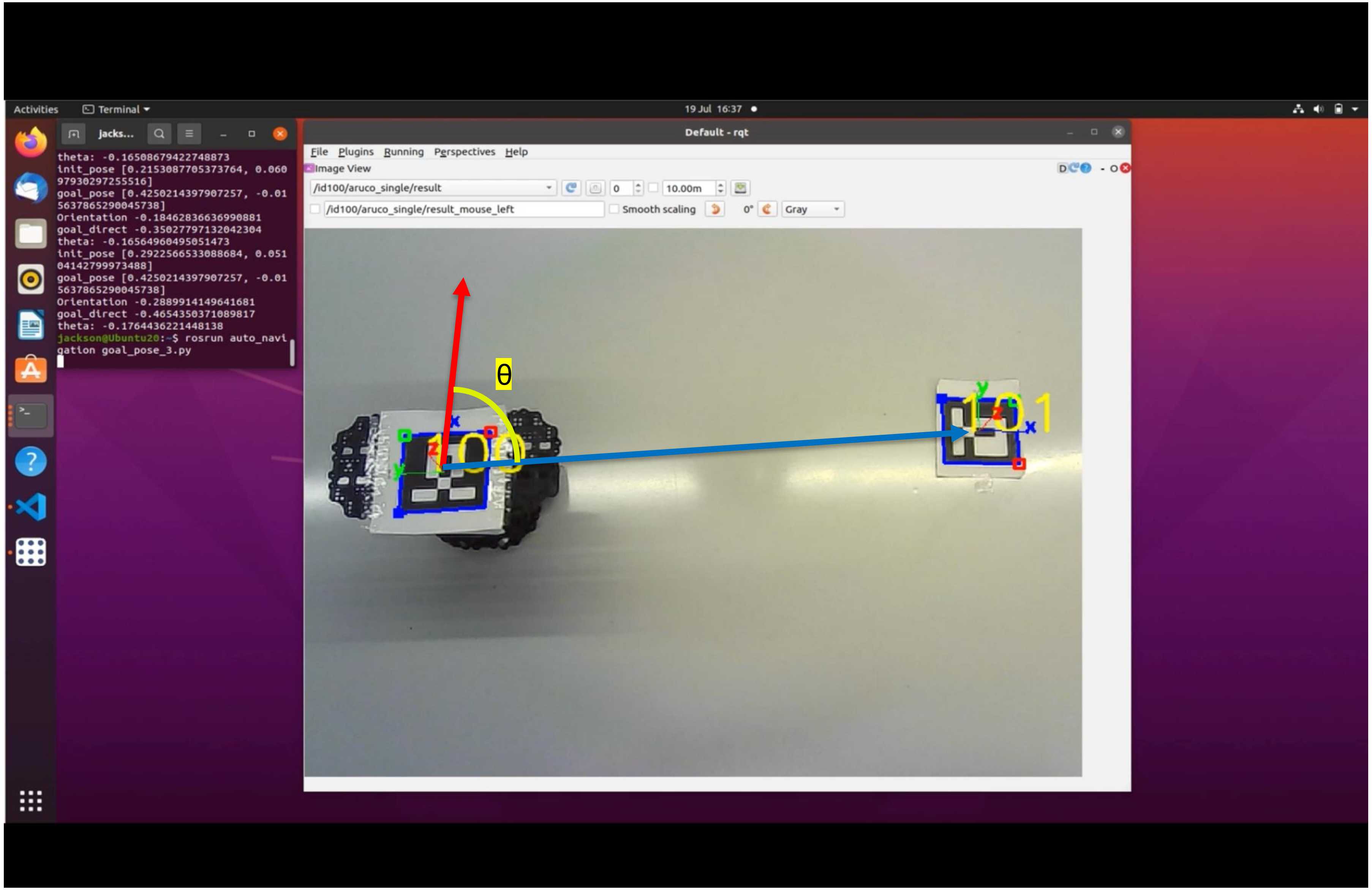
- Refer to the manual for environment setup and instructions.

Motion control for one turtlebot

- Angular velocity
  - ❑ Kp_angular * theta
- Linear velocity
  - ❑ Kp_linear * math.cos(theta)



Motion planning for single real robot



Current orientation

Goal direction

ROS publisher

- ROS tutorial: http://wiki.ros.org/ROS/Tutorials

- Example cmd_vel publisher
  - ❑ Set linear velocity to 0.1m/s
  - ❑ Set angular velocity to 0.

```python
#!/usr/bin/env python
# license removed for brevity
import rospy
from geometry_msgs.msg import Twist

def vel_publisher():
        pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
        rospy.init_node(‘vel_publisher', anonymous=True)
        twist = Twist()
        twist.linear.x = 0.1
        twist.angular.z = 0.0
        pub.publish(twist)

if __name__ == '__main__’:
        try:
            vel_publisher()
        except rospy.ROSInterruptException:
            pass
```

Motion planning for multiple turtlebots

- Namespace: namespace is a unique identifier for each turtlebot.

- During bringup for the first robot(tb3_0), run:

```
$ ROS_NAMESPACE=tb3_0 roslaunch turtlebot3_bringup turtlebot3_robot.launch
multi_robot_name:="tb3_0" set_lidar_frame_id:="tb3_0/base_scan"
```

- During bringup for the second robot(tb3_1), run:

```
$ ROS_NAMESPACE=tb3_1 roslaunch turtlebot3_bringup turtlebot3_robot.launch
multi_robot_name:="tb3_1" set_lidar_frame_id:="tb3_1/base_scan"
```

- In this case, the original topic name like ‘cmd_vel’ will change to ‘tb3_0/cmd_vel’ for robot tb3_0 and ‘tb3_1/cmd_vel’ to robot tb3_1.

Motion control for one turtlebot

- In **multi_robot_navigation_deliver.py**

```
distance = math.dist(current, next)

k1 = 5
k2 = 20

A=20
linear =min(A*math.exp(k1 * distance), 30.0)
angular = k2 * theta

rightWheelVelocity = linear + angular
leftWheelVelocity = linear - angular
```

Motion planning for multiple turtlebots

- Motion control:
  - Similar motion control algorithm for single-robot motion control
  - Velocity of tb3_0 needs to be published to the topic ‘tb3_0/cmd_vel’
  - Velocity of tb3_1 needs to be published to the topic ‘tb3_1/cmd_vel’