

Functional Implementation

The goal of an autonomous system is to operate within an environment without human interaction. The system needs to be able to understand itself and the world around it in order to determine which path to take and what the right commands are to get the system to follow that path. We can divide these autonomous capabilities into four main areas: sense, perceive, plan, and act.

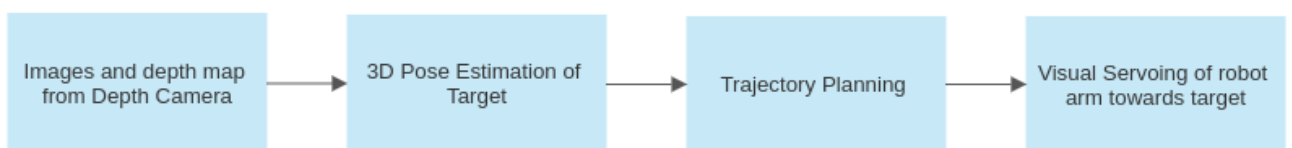


Sense: Depth Camera (RGBD Camera)

Perceive: YOLO v8 Small object detection model, Inverse Projection Transformation, Coordinate transformations & 3D Pose Estimation of the target object using tf2_ros library, Joint states information from arm_controller

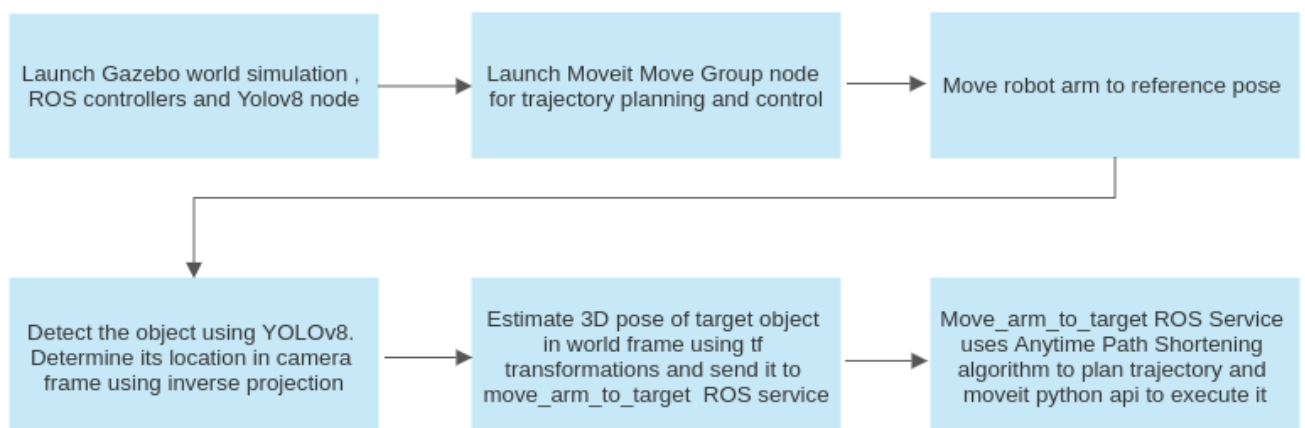
Decide & Plan: Anytime Path Shortening (default planners: RRT Connect, RRT, TRRT, BKPIECE) planner from OMPL Library

Act: moveit_commander Python package from Moveit and ROS



The control flow of the code is visually depicted in the provided diagram, illustrating the sequence of operations :

CONTROL FLOWCHART



Technical Details

- **Directory Structure :**

```
raghu@raghu-IdeaPad-S-15ALC05:~/Object_follower_UR5/src$ tree -L 1
.
├── arm_moveit_config
├── CMakeLists.txt -> /opt/ros/noetic/share/catkin/cmake/toplevel.cmake
├── dh_gripper_msgs
├── dh_robotics_ag95_gripper
├── models
├── robot_arm
├── shell_scripts
├── ur_description
├── ur_gazebo
└── yolov8_ros

9 directories, 1 file
```

ur_description - contains meshes (stl and dae files), urdf, and xacro files of UR5 robotic arm. Taken mainly from the Universal Robots GitHub repository. The main edited file is ur_macro.xacro file.

ur_gazebo - contains launch files to launch UR5 arm in gazebo simulation. Taken mainly from the Universal Robots Github repository. The main files in this folder are the ur_control.launch.xml file (in /launch/inc folder) and the ur5_controllers.yaml file (in /cfg folder)

dh_robotics_ag95_gripper and **dh_gripper_msgs** - contains files related to DH Robotics AG95 gripper. Provided by mentor Krrish Jindal. Also available in the DH Robotics GitHub repository. It has meshes, urdf, xacro, and gazebo plugin files of the gripper.

shell_scripts - contains shell scripts for launching multiple roslaunch files easily at one command. The main files are object_follower_ur5.sh file and pbvs.sh file. Instructions for use are already present in the README.md file.

models - has gazebo models of boxes

yolov8_ros - ROS1 wrapper for YOLO v8. ROS package for enabling the yolov8 model from Ultralytics library to run on images obtained from camera in gazebo simulation. Also has code for Inverse Depth Projection and hence publishes the location of detected objects wrt. camera frame. The main file is yolo_ros.py

arm_moveit_config - Moveit configuration files for the robot arm. Mainly generated automatically by moveit setup assistant. The main files are ur5_robot.srdf file, ros_controllers.yaml file, ompl_planning.yaml file, demo.launch file, move_group.launch file, trajectory_execution.launch.xml file and moveit_planning_execution.launch file

robot_arm - Has all important scripts of tf transforms, 3D Pose estimation, move arm ROS Client node, move arm to target ROS Service node, and other test nodes. It also has the majority of important launch files and also has service files.

- **Scripts**

Main python scripts are files contained in the robot_arm/scripts directory and yolo_ros.py from the yolov8_ros ROS package.

- **yolov8_ros ROS package/wrapper:**

ROS1 wrapper or package for using YOLOv8 from the Ultralytics library was developed. The original version used only the pre-trained yolov8 nano model. The yolo_ros.py python file was edited to allow using custom weights. Also, code for inverse projection transformation was added to the script. A new rostopic for publishing locations of detected objects (wrt. camera frame) was made. For this purpose, some new message files were added. These edits are now included in the original repository as my pull request was merged, and I too have become a contributor. Original repository: https://github.com/TPODAvia/yolov8_ros

Fork on which I worked: https://github.com/Vamsi-IITI/yolov8_ros

My commits and pull request can be seen through below links:

https://github.com/Vamsi-IITI/yolov8_ros/commits/master

https://github.com/TPODAvia/yolov8_ros/commit/ddca79ff08bc958c9654f941edb25fc936215486

This package can be used in various projects. One needs to just provide their custom weights and a text file containing the names of classes. Instructions for use are already available on GitHub.

In this project, this package helps detect boxes in gazebo simulation and publish their location wrt. camera frame to /yolov8/ObjectLocation rostopic.

Subscribed Topics:

- [img_topic](#) (sensor_msgs/Image)

img_topic mentioned in the launch file.
Here: /camera/rgb/image_raw

- [depth_topic](#) (sensor_msgs/Image)

depth_topic mentioned in the launch file.
Here: /camera/depth/image_raw

- [camera/depth/camera_info](#) (sensor_msgs/CameraInfo)
to extract camera intrinsic parameters

Published Topics:

- [/yolov8/BoundingBoxes](#) - (yolov8_ros_msgs/BoundingBoxes)

Publishes bounding box-related information (xmin, xmax, ymin, ymax) along with class and probability of detected objects

- [/yolov8/DepthPoints](#) - (yolov8_ros_msgs/DepthPoints)

Publishes depth/distance of the center of detected objects from the camera. It also publishes (x,y) coordinates of the absolute center and offset center of the bounding box, along with class and probability

- [/yolov8/ObjectLocation](#) - (yolov8_ros_msgs/ObjectLocations)

Publishes the location of detected objects with respect to the camera frame, along with the class and probability of detected objects

- [/yolo_visualization](#) - (sensor_msgs/Image)

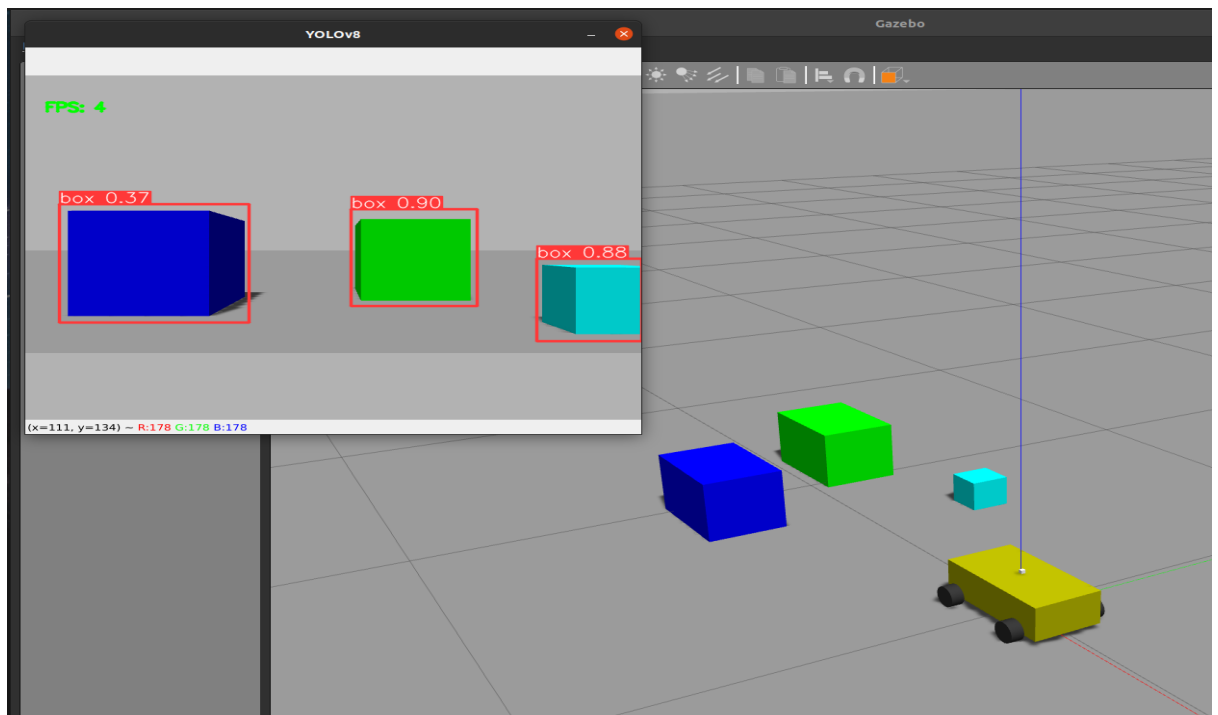
For visualizing the results of Yolo object detection in rviz

Parameters:

There are six ROS Parameters that can be used to configure the yolov8_node

- Path to trained weights
~weights_path (str, default: "yolov8n.pt")
- Path to text file with classes
~classes_path (str, default: "\$(find yolov8_ros)/model/coco.txt")
- Image topic
~img_topic (str, default: "/camera/rgb/image_raw")
- Depth Image topic
~depth_topic (str, default: "/camera/depth/image_raw")
- Queue size
~queue_size (int, default: 1)
- Visualization of results
~visualize (bool, default: False)

For testing purposes, the package also has sim_yolo package which has sim_yolo_demo.launch file which launches a gazebo world simulation with AGV (equipped with a depth camera) in it. It also launches the yolo_ros node. Hence objects can be inserted in the Gazebo simulation in front of the camera, and trained custom weights can be tested. Gazebo models of boxes can be taken from models directory.



```
raghu@raghu-IdeaPad-5-15ALC05:~/Object_follower_UR5/src/yolov8_ros$ tree -L 3
.
├── ReadMe.md
├── sim_yolo
│   ├── CMakeLists.txt
│   ├── launch
│   │   └── sim_yolo_demo.launch
│   ├── models
│   │   └── tank
│   └── package.xml
├── yolov8_ros
│   ├── CMakeLists.txt
│   ├── launch
│   │   └── yolov8.launch
│   ├── model
│   │   ├── berkeley.txt
│   │   ├── best.pt
│   │   ├── box.txt
│   │   └── coco.txt
│   ├── package.xml
│   ├── scripts
│   │   └── yolo_ros.py
├── yolov8_ros_msgs
│   ├── CMakeLists.txt
│   ├── msg
│   │   ├── BoundingBoxes.msg
│   │   ├── BoundingBox.msg
│   │   ├── DepthPoint.msg
│   │   ├── DepthPoints.msg
│   │   ├── ObjectLocations.msg
│   │   └── Object.msg
│   └── package.xml
└── 10 directories, 20 files
```

- **robot_arm ROS Package:**

This is the main package that brings everything together. It has important launch files and Python scripts for 3D pose estimation, Robot arm manipulation, ROS client and service nodes, etc. Also has service files and world files.

Directory Structure:

```
raghu@raghu-IdeaPad-5-15ALC05:~/Object_follower_UR5/src/robot_arm$ tree -L 2
.
├── CMakeLists.txt
├── launch
│   ├── box_world.launch
│   ├── box_yolo.launch
│   ├── go_to_referencepos.launch
│   ├── move_arm_client.launch
│   ├── pbvs.launch
│   ├── robot_arm_manipulation.launch
│   └── ur5_empty_world.launch
├── package.xml
├── scripts
│   ├── arm_manipulation.py
│   ├── cartesian_path_planner.py
│   ├── go_to_referencepos.py
│   ├── move_arm_client.py
│   ├── object_location_converter_node.py
│   ├── pbvs.py
│   ├── robot_arm_manipulation.py
│   └── rrt_path_planner.py
├── srv
│   └── target_pose.srv
├── urdf
│   └── ur5.xacro
├── worlds
│   ├── box.world
│   └── empty.world
└── 5 directories, 21 files
```

Scripts:

1. `arm_manipulation.py` uses the `moveit_commander` Python package and the “Anytime Path Shortening” planner from OMPL to plan and execute trajectory to a hard-coded target
2. `cartesian_path_planner.py` uses the `moveit_commander` Python package and the `compute_cartesian_path` function to plan and execute a cartesian trajectory between the start and goal position
3. `go_to_referencepos.py` uses the `moveit_commander` Python package and the “Anytime Path Shortening” planner from OMPL to plan and execute trajectory to move the arm to a reference pose (which is set in `ur5_robot.srdf` file present in the moveit configuration files of the robot arm)

4. `move_arm_client.py` selects the detected object with the highest probability, then uses `tf2` and `tf2_ros` packages for converting its coordinates from camera frame to world frame, and then it uses the target location and calls `move_arm_to_target` service. ROS Client node.
5. `object_location_convertor_node.py` selects the detected object with the highest probability, then uses `tf2` and `tf2_ros` packages for converting its coordinates from the camera frame to the world frame. It prints the target location in the camera frame and in the world frame on the screen. `move_arm_client.p` file was derived from this. This file is useful for debugging purposes also.
6. `pbvs.py` is the ideal case code for arm manipulation to a specific point. It takes the target from user input and then uses the “Anytime Path Shortening” planner from OMPL to plan trajectory and the `moveit_commander` package to move the robot arm to the target
7. `robot_arm_manipulation.py` starts `move_arm_to_target` ROS Service. It receives a target, then uses the `moveit_commander` Python package and the “Anytime Path Shortening” planner from OMPL to plan and execute trajectory to move the arm to a target pose. ROS Service node.
8. `rrt_path_planner.py` uses the `moveit_commander` Python package and the “RRTConnect” planner from OMPL to plan and execute trajectory to a hard-coded target. This was the file from which `arm_manipulation.py` was derived.

Launch files:

1. `box_world.launch` launches robot arm in the `box.world` gazebo environment, launches `robot_state_publisher`, controllers using `ur_control.launch.xml` and `box_yolo.launch` file too.
2. `box_yolo.launch` starts the `yolo_ros` node with custom-trained YOLOv8 Small model weights capable of detecting boxes
3. `go_to_reference_pos.launch` file launches `go_to_reference_pos.py` script
4. `move_arm_client.launch` file launches `move_arm_client.py` script
5. `pbvs.launch` file launches `pbvs.py` script
6. `robot_arm_manipulation.launch` file launches `robot_arm_manipulation.py` script
7. `ur5_empty_world.launch` file launches robot arm in empty world environment with `robot_state_publisher` and controllers. It is for the Robot Visualization task

Services:

target_pose.srv was made for move_arm_to_target service. It has a simple structure. Its request field takes the x,y, and z coordinates of the target while the response is a String message.

```
raghu@raghu-IdeaPad-5-15ALC05:~/Object_follower_UR5$ rossrv show target_pose
[robot_arm/target_pose]:
float64 x
float64 y
float64 z
---
string message
```

- **arm_moveit_config (Moveit configuration files for robot arm)**

This directory contains the moveit configuration files of the robot arm. Mainly generated automatically by moveit setup assistant. The main files are ur5_robot.srdf file, ros_controllers.yaml file, ompl_planning.yaml file, demo.launch file, move_group.launch file, trajectory_execution.launch.xml file and moveit_planning_execution.launch file .

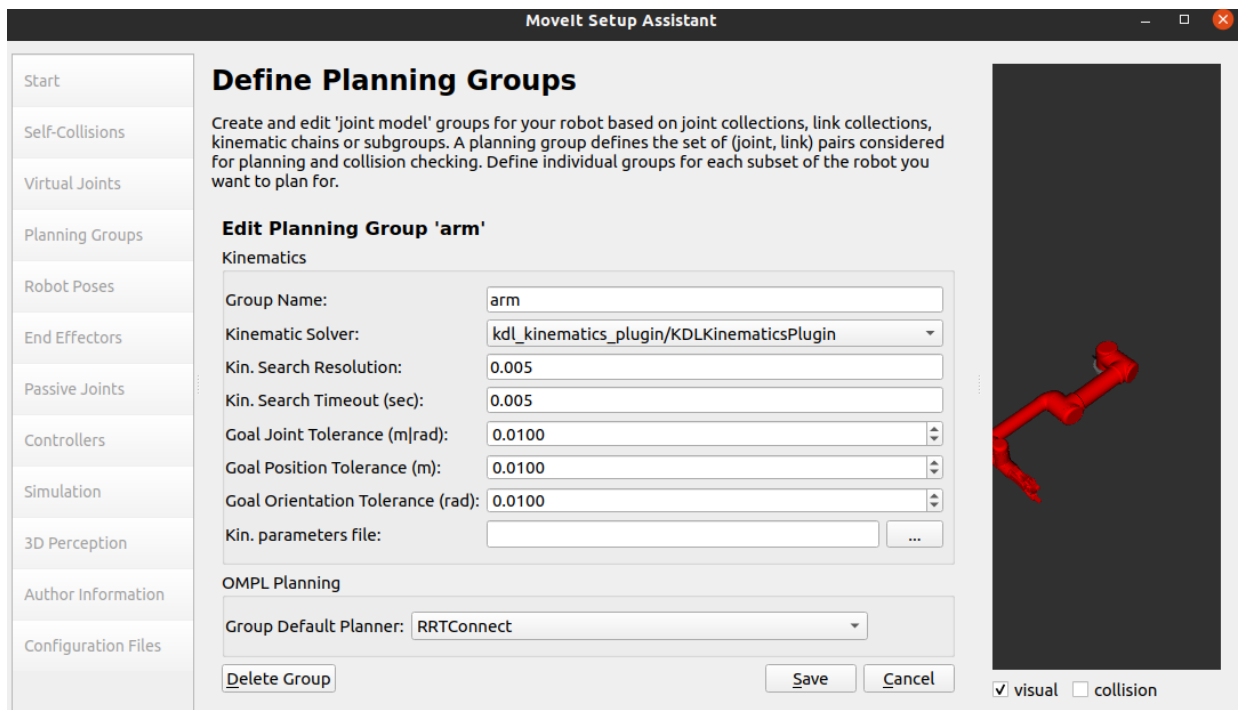
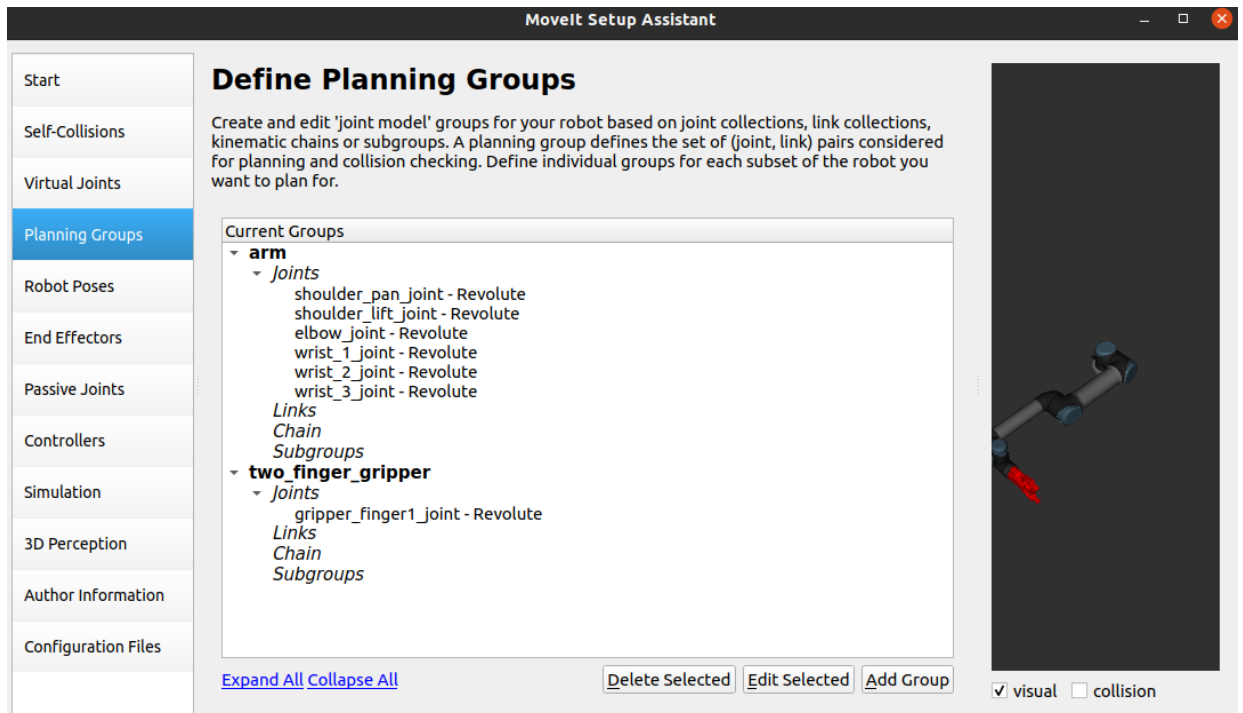
Directory Structure:

```
raghu@raghu-IdeaPad-5-15ALC05:~/Object_follower_UR5/src/arm_moveit_config$ tree -L 2
.
├── CMakeLists.txt
├── config
│   ├── cartesian_limits.yaml
│   ├── chomp_planning.yaml
│   ├── fake_controllers.yaml
│   ├── gazebo_controllers.yaml
│   ├── gazebo_ur5_robot.urdf
│   ├── joint_limits.yaml
│   ├── kinematics.yaml
│   ├── ompl_planning.yaml
│   ├── ros_controllers.yaml
│   ├── sensors_3d.yaml
│   ├── simple_moveit_controllers.yaml
│   ├── stomp_planning.yaml
│   └── ur5_robot.srdf
├── launch
│   ├── chomp_planning_pipeline.launch.xml
│   ├── default_warehouse_db.launch
│   ├── demo_gazebo.launch
│   ├── demo.launch
│   ├── fake_moveit_controller_manager.launch.xml
│   ├── gazebo.launch
│   ├── joystick_control.launch
│   ├── move_group.launch
│   ├── moveit_planning_execution.launch
│   ├── moveit.rviz
│   ├── moveit_rviz.launch
│   ├── ompl-chomp_planning_pipeline.launch.xml
│   ├── ompl_planning_pipeline.launch.xml
│   ├── pilz_industrial_motion_planner_planning_pipeline.launch.xml
│   ├── planning_context.launch
│   ├── planning_pipeline.launch.xml
│   ├── ros_controllers.launch
│   ├── ros_control_moveit_controller_manager.launch.xml
│   ├── run_benchmark_ompl.launch
│   ├── sensor_manager.launch.xml
│   ├── setup_assistant.launch
│   ├── simple_moveit_controller_manager.launch.xml
│   ├── stomp_planning_pipeline.launch.xml
│   ├── trajectory_execution.launch.xml
│   ├── ur5_robot_moveit_sensor_manager.launch.xml
│   ├── warehouse.launch
│   └── warehouse_settings.launch.xml
└── package.xml

2 directories, 42 files
```

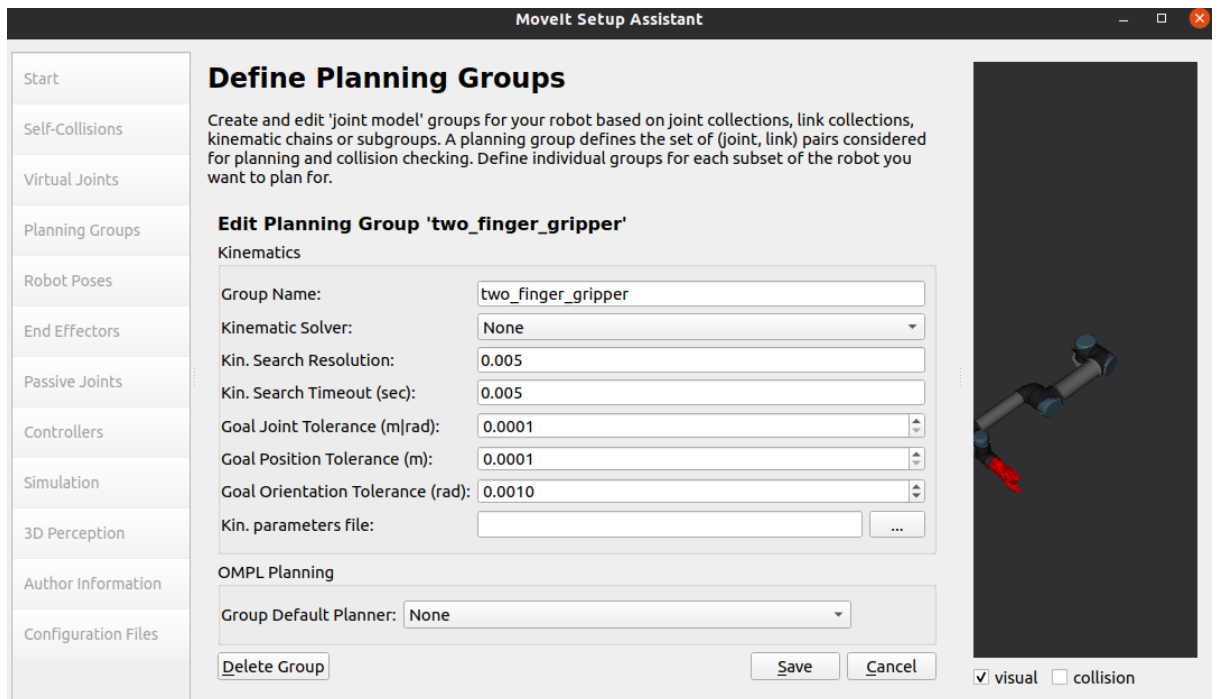

Moveit configuration information:

1. Planning groups

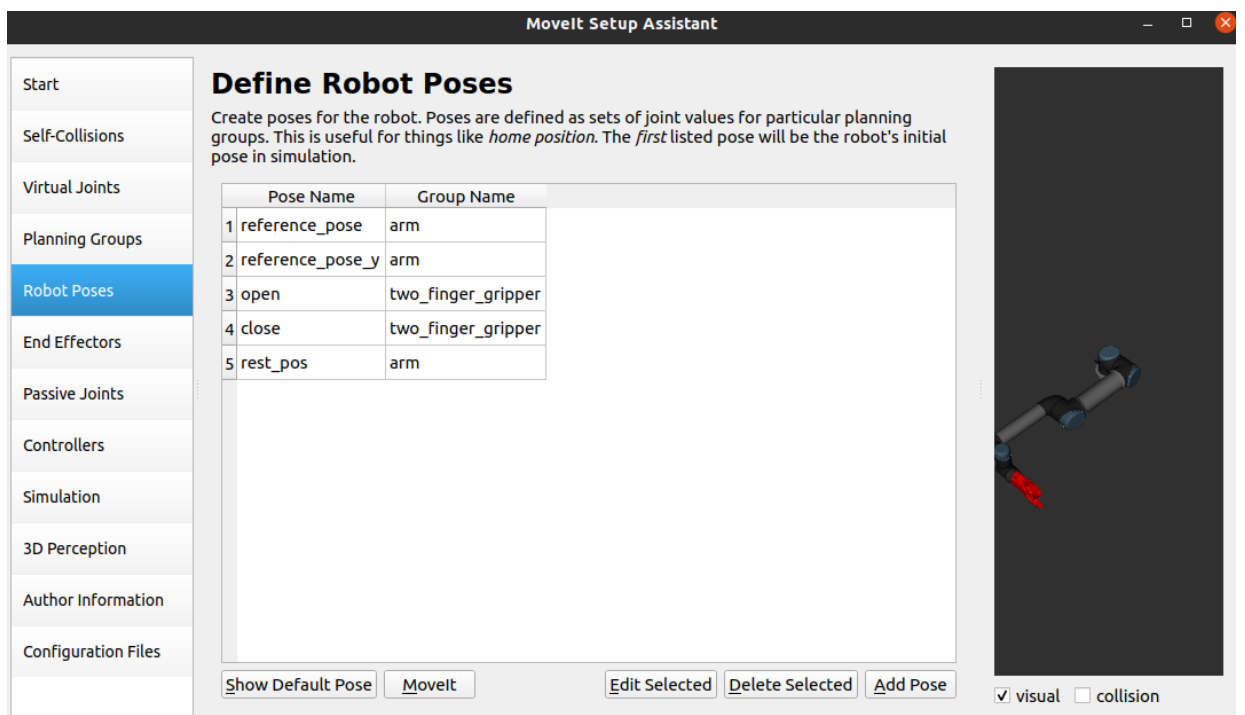


Default Kinematic Solver: **KDL** ([Kinematic and Dynamics Library - Orocos Project](#))

OMPL Default Planner: **RRTConnect**



2. **Robot poses** (more can be added either by editing ur5_robot.srdf or using the moveit setup assistant)



3. End Effectors

Start

Self-Collisions

Virtual Joints

Planning Groups

Robot Poses

End Effectors

Passive Joints

Controllers

Simulation

3D Perception

Author Information

Configuration Files

Define End Effectors

Setup your robot's end effectors. These are planning groups corresponding to grippers or tools, attached to a parent planning group (an arm). The specified parent link is used as the reference frame for IK attempts.

	End Effector Name	Group Name	Parent Link	Parent Group
1	two_finger_gripper	two_finger_gripper	wrist_3_link	arm

Edit Selected

Delete Selected

Add End Effector



☒ visual ☐ collision

4. Passive Joints

Start

Self-Collisions

Virtual Joints

Planning Groups

Robot Poses

End Effectors

Passive Joints

Controllers

Simulation

3D Perception

Author Information

Configuration Files

Define Passive Joints


Specify the set of passive joints (not actuated). Joint state is not expected to be published for these joints.

Active Joints

	Joint Names
1	shoulder_pan_joint
2	shoulder_lift_joint
3	elbow_joint
4	wrist_1_joint
5	wrist_2_joint
6	wrist_3_joint
7	gripper_finger1_inner_knuckle_joint
8	gripper_finger1_finger_tip_joint
9	gripper_finger1_joint
10	gripper_finger1_finger_joint
11	gripper_finger2_inner_knuckle_joint
12	gripper_finger2_finger_tip_joint
13	gripper_finger2_joint

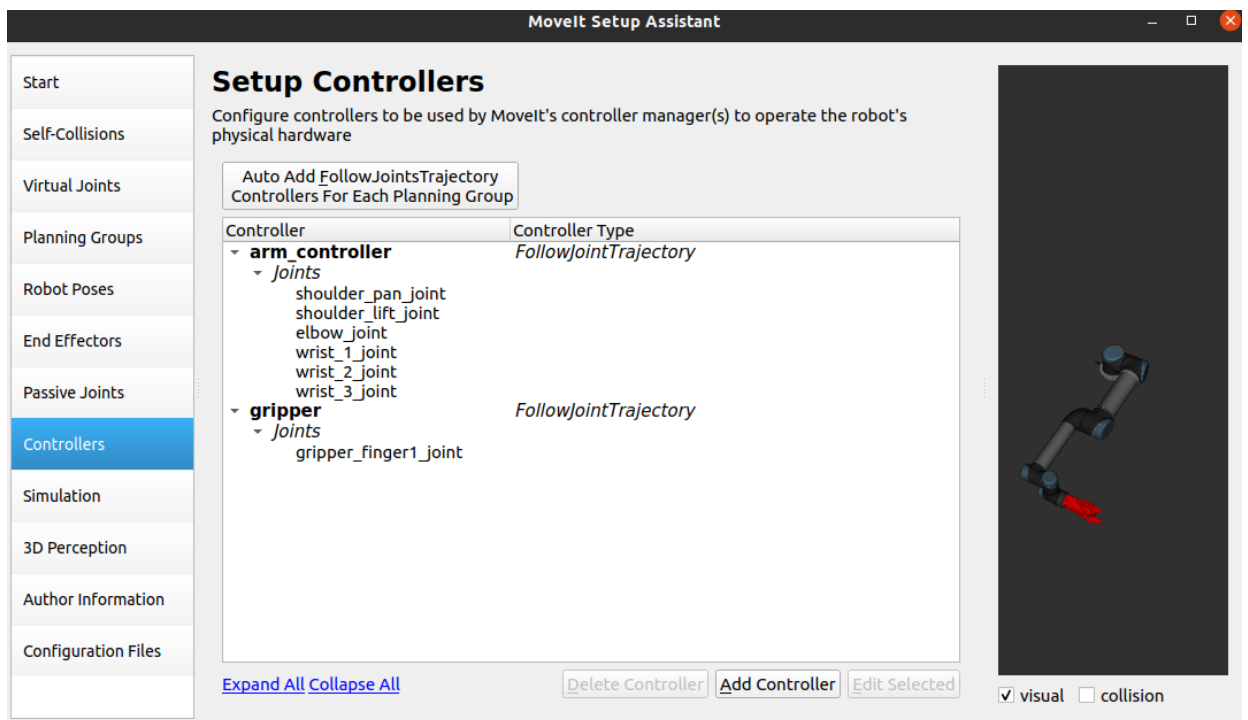
Passive Joints

	Joint Names
1	gripper_finger1_inner_knuckle_joint
2	gripper_finger1_finger_tip_joint
3	gripper_finger1_finger_joint
4	gripper_finger2_inner_knuckle_joint
5	gripper_finger2_finger_tip_joint
6	gripper_finger2_joint
7	gripper_finger2_finger_joint



☒ visual ☐ collision

5. Controllers



- **Note:**

1. Auto-generated `ros_controllers.yaml` file contains wrong information. You have to correct it according to your robot arm. For this project, it was corrected by me.
2. Select joints in the planning groups and passive joints carefully.
3. You can try a variety of Kinematic Solvers like **KDL**, **TracIK**, and IKFast, or you can even use kinematic solver plugins from Universal Robots. For that, you need to download the folder `ur_kinematics` from Universal Robots GitHub Repository. [Here](#)
4. You can select any Default planner from OMPL. You don't need to worry about it as the `set_planner_id` method of the `moveit_commander` package allows you to use your desired planner in place of the default planner.
5. STOMP or CHOMP planners can be added in pre-processing or post-processing stage of trajectory planning by OMPL. Planning Request Adapters is a concept in MoveIt that can be used to modify the trajectory (pre-processing and/or post-processing) for a motion planner. Using the concepts of Planning Adapters, multiple motion planning algorithms can be used in a pipeline to produce robust motion plans. For example, a sample pipeline of motion plans might include an initial plan produced by OMPL which can then be optimized by CHOMP to produce a motion plan which would likely be better than a path produced by OMPL or CHOMP alone.

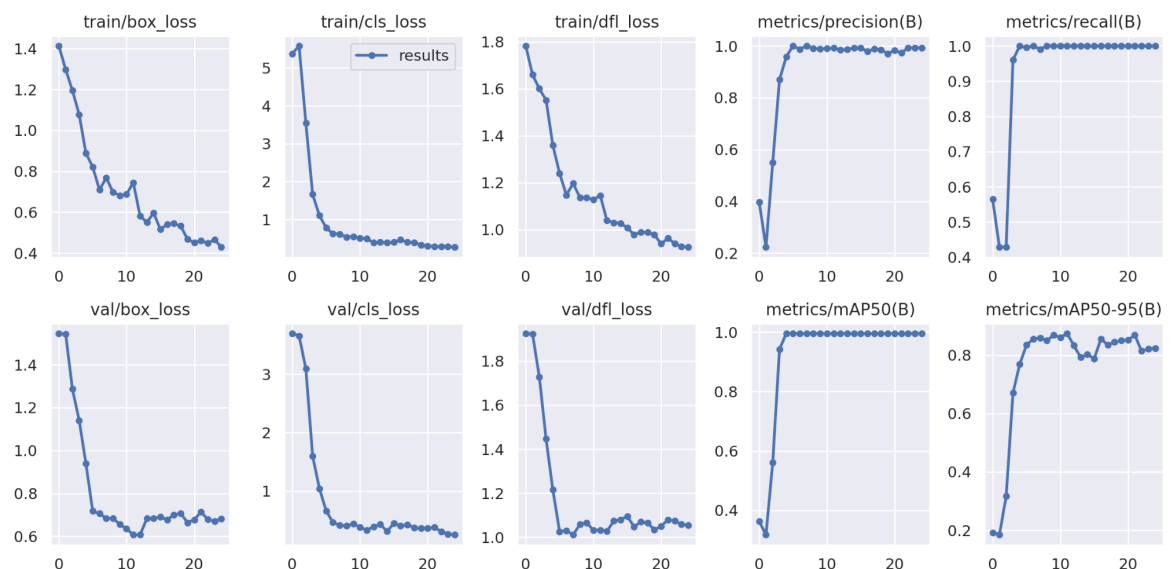
6. moveit_planning_execution.launch is not auto-generated. It was created by me.

● Yolov8 Training

Trained Yolo v8 Small model (Yolo v8s ~ 20 MB) for object detection and tracking. Here the selected object I took is a box. Though Yolo v8l and v8x are more accurate but due to their enormous size, they have lower FPS rates. Hence I selected Yolo v8 Small and achieved a decent FPS of 3-5 with an average inference time being approx 200 milliseconds. So there is a trade-off between accuracy and speed.

Training results : (Trained over 25 epochs , best weights were chosen for use in project)

Github Repo : https://github.com/Vamsi-IITI/box_yolov8



● Issues

1. Currently, the 3D pose estimation of the target has slight errors which makes it hard to achieve object tracking and closed-loop feedback control. Currently, it is an open-loop system.

Status: Not Resolved ❌

Consequence: The fifth assigned task hasn't been achieved ▼

Object tracking of moving object, it's motion prediction, target pose update, and closed feedback system for visual servoing of arm still remains

Alternate Solution: Instead of using an eye-in-hand configuration of the robot arm, we can fix the camera at the base of the robot arm or any other stationary place. Improving the accuracy of Inverse Projection Mapping. Or usage of aruco tags / RFID tags for better 3D pose estimation of target

2. Yolov3, as well as Yolov3 Tiny weights, didn't work when they were used with the darknet_ros package in the gazebo simulation. Yolov3 trained weights had large size too and a very low FPS rate of 0.1

Repo: [Here](#)


Status: Resolved 

Solution: Migrated to the latest state-of-the-art computer vision model Yolo v8 provided by Ultralytics Library. Developed a ROS wrapper for utilizing it.

Repo: https://github.com/Vamsi-IITI/yolov8_ros

3. Sometimes start position tolerance is violated and moveit says CONTROL ABORTED. Even after increasing the start position tolerance in trajectory_execution.launch.xml didn't help. But the good thing is that robot_arm executes the trajectory still.

Status: Not Resolved

Comment: Not a major issue, just make sure whether the desired result is obtained or not. If not, then please re-run the simulation/script. 

4. Haven't tried Planning Request Adapters for Trajectory Optimization
5. Need to add Collision checking with Bullet Library. Also, need to utilize the octomap service of Moveit by configuring sensors

Final Deliverables

The final deliverables of the project are code/scripts for visual servoing and perception along with this report.

GitHub Repository: https://github.com/Vamsi-IITI/Object_follower_UR5/tree/main

Subsidiary Repositories:-

1. yolov8_ros package: https://github.com/Vamsi-IITI/yolov8_ros.git (Can be used in other projects)
2. box_yolov8 repo: https://github.com/Vamsi-IITI/box_yolov8 (Yolov8 Training notebook, weights, results, and dataset. Scripts for object detection in image and object tracking in the video)

References

Many resources were used while developing this project. Every effort has been made to acknowledge and incorporate all the sources that contributed to the successful completion of this endeavor. It is my sincere hope that this comprehensive documentation will prove beneficial to others.

- ROS Wiki (<http://wiki.ros.org/ROS/Tutorials>)
- Universal Robotics ([github ros-industrial/universal Robot](https://github.com/ros-industrial/universal_robot)) (For UR5 robot arm related files)
- DH Robotics AG 95 Gripper (https://github.com/DH-Robotics/dh_gripper_ros) (For gripper files)
- Ultralytics (<https://github.com/ultralytics/ultralytics.git>) (For YOLOv8)
- OMPL Planners (<https://ompl.kavrakilab.org/planners.html>)
- YOLOv8 ROS package (https://github.com/TPODAvia/yolov8_ros) (Added features in original repo which are now merged . I am now a contributor of this repo)
- Inverse Projection Transformation ([Inverse Projection Transformation | by Daryl Tan | Towards Data Science](#)) (Great article!)
- MoveIt Tutorials ([Getting Started — moveit_tutorials Noetic documentation](#))
- MoveIt Training for Beginners , By Construct ([MoveIt Training for Beginners](#))
- MoveIt! Robot Manipulators , By Robotics with Sakshay ([MoveIt! Robot Manipulators | Part - 5 | ROS Learning Series](#))
- ROS Industrial Training ([Application Demo 1 - Perception-Driven Manipulation — Industrial Training documentation](#))
- Move Group Commander Class Reference (http://docs.ros.org/en/jade/api/moveit_commander/html/classmoveit__commander_1_1move_group_1_1MoveGroupCommander.html)
- tf2 Tutorials (<http://wiki.ros.org/tf2/Tutorials>)
- tf2 ROS example (<https://gist.github.com/ravijo/cb560eeb1b514a13dc899aef5e6300c0>)
- Roboflow YOLOv8 Colab Notebook ([train-yolov8-object-detection-on-custom-dataset.ipynb - Colaboratory](#))
- YOLO v8 Box Detection (https://github.com/Vamsi-IITI/box_yolov8.git) (For YOLO v8s model , dataset and results along with some python scripts)

- UR5-ROS-Gazebo (https://github.com/lihuang3/ur5_ROS-Gazebo.git) (For cartesian path planning code)
- Custom Manipulator Simulation in Gazebo and Motion Planning with MoveIt! ([Custom Manipulator Simulation in Gazebo and Motion Planning with MoveIt! | by Tahsincan Kose | Medium](#)) (A really nice article)
- Fetch Robotics Tutorials (<https://docs.fetchrobotics.com/manipulation.html>)
- Foxglove ROS1 Tutorials ([Simulating Robotic Scenarios with ROS 1 and Gazebo - Foxglove](#))
- Yolo Darknet Guide ([YOLOv4-darknet installation and usage on your system \(Windows & Linux\) | by Techzizou | Geek Culture | Medium](#))
- A Gentle Introduction to YOLO v4 for Object detection in Ubuntu 20.04 ([A Gentle Introduction to YOLO v4 for Object detection in Ubuntu 20.04](#))
- Darknet ROS (https://github.com/leggedrobotics/darknet_ros)
- YOLO Object Detection | ROS Developers Live Class , By Construct (<https://www.youtube.com/live/dB0Sijo0RLs?feature=share>)
- Coordinates and Transformations ([CoordinateTransformations](#))
- [Position-based visual servo control of autonomous robotic manipulators - ScienceDirect](#)
- [A tutorial on visual servo control | IEEE Journals & Magazine](#)
- [Characteristics based visual servo for 6DOF robot arm control - ScienceDirect](#)
- [Handbook of Robotics Chapter 24: Visual servoing and visual tracking](#)
- [Vision-based control of UR5 robot to track a moving object under occlusion using Adaptive Kalman Filter | Proceedings of the Advances in Robotics 2019](#)
- [Visual servoing - Wikipedia](#)

Thank You!

Raghuvamsi Bokka
 B.Tech Mechanical Engg.
 IIT Indore (2020-2024)
 Email: raghuvamsibokka5@gmail.com
 Date: 07 / 08 / 2023