Pluto Project

Authors:

Dina Svetlitsky dinasv@post.bgu.ac.il

Alex Kovalchuk alexds9@gmail.com

Revision: #1 (2016.03.01)

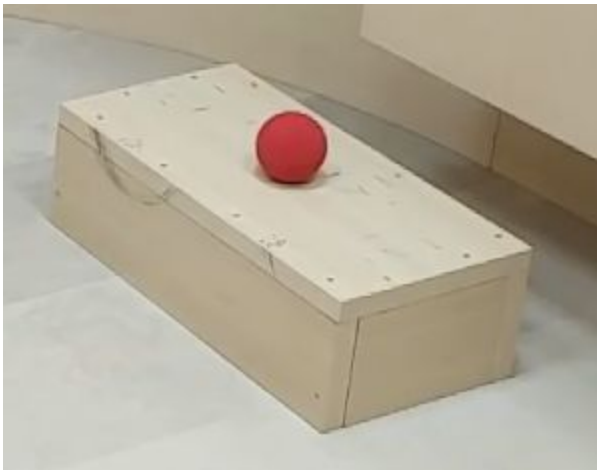
Table of content:

1. Introduction

a. Komodo robot with elevator by RoboTiCan

b. Red non-glossy ball
   i. Weight: 8 gram
   ii. Diameter: 0.075 meter

c. Bench
   i. Height: 0.25 meter



The robot is placed on a flat floor. With a single red object: red ball - placed on a bench (tested in the distance below 5 meters from the robot). Assuming no obstacles between robot and the ball. Robot should find the ball, approach it, and lift it with an arm.

Example of the initial state:

Example of the target state:

2. Calibrations

a. Asus (top) camera

Camera angle should be adjusted by placing the red ball at 1.6 meter distance in front of the robot, on the floor, so it would fully appear just above the bottom edge, in the image from top camera. If the ball is placed closer than 1.6 meter, it should disappear (partially or completely) from camera view.

b. Elevator

Arm should be placed at the lowest possible position beforehand. (May take up to 10 minutes to lower from top most position.)

c. Arm camera

Arm camera has a sporadic issue with color distribution after HW boot, red color appear as green. If this issue appear, arm HW module should be power cycled.

3. Interfaces with Komodo

The following ROS topics function as interfaces to access hardware modules of Komodo robot.

a. /Asus_Camera/rgb/image_raw
   i. Image from Asus (top) camera.
   ii. Resolution: 640x480 px.
b. /Creative_Camera/rgb/image_raw
   i. Image from arm camera.
   ii. Resolution: 320x240 px.
   iii. Left and right are swapped.
   iv. Sporadic color distribution issue.
c. /diff_driver/command
   i. Requests for HW movement module (wheels motors).
   ii. Requests should be repeated, for persistent motion.
   iii. Control linear and angular movement.
d. /diff_driver/odometry
   i. Supposed to report exact movement made by the robot in linear and angular directions in meters.
   ii. Found unreliable, possibly not calibrated, both for linear and angular movement.
e. /left_urf
   i. Sonar proximity sensor on the left side of the robot.
   ii. Has a tendency for false readings.
f. /right_urf
   i. Sonar proximity sensor on the right side of the robot.

ii. Has a tendency for false readings.

g. /scan
   i. Laser scanner in the front of the robot.
   ii. Used to detect obstacles by finding the minimum distance, in the array of distances, in last scan.

h. /[joint]_controller/command
   i. Request for moving the arm joint in radians.
   ii. Requests should be repeated, for persistent motion.
      1. /base_rotation_controller/command
      2. /shoulder_controller/command
      3. /elbow1_controller/command
      4. /elbow2_controller/command
      5. /wrist_controller/command
      6. /left_finger_controller/command
      7. /right_finger_controller/command

i. /[joint]_controller/state
   i. Returns a joint position with a certain error
      1. /base_rotation_controller/state
      2. /shoulder_controller/state
      3. /elbow1_controller/state
      4. /elbow2_controller/state
      5. /wrist_controller/state
      6. /left_finger_controller/state
      7. /right_finger_controller/state

4. Algorithm

init close_range = false, using_top_camera = true

main_loop:

1. if true == using_top_camera
    1.1. Scan the image from top camera for the red ball
2. else Scan the image from arm camera for the red ball
3. if ball found
    3.1. Turn the robot until it is in front of the ball
    3.2. if true == using_top_camera
        3.2.1. close_range = is ball place at the picture looks closer than 2.2m (lower than certain Y-axis value)
        3.2.2. Make a step toward the ball
    3.3. else
        3.3.1. if in ~0.5m distance (by ball radius) or less
            3.3.1.1. call ball_catcher
        3.3.2. else
            3.3.2.1. Make a step toward the ball
4. else
    4.1. if false == close_range
        4.1.1. make a step in clockwise direction
    4.2. else
        4.2.1. using_top_camera = true
        4.2.2. Set arm to scanning position

ball_catcher:

5. if ball close enough to pick it
    5.1. Move the arm closer

5.2. Grab the ball
5.3. Lift the ball

6. else
   6.1. move the arm base left or right to make the ball at horizontal center for arm camera
   6.2. move the elbow_2 to make the ball at vertical center for arm camera
   6.3. move the arm shoulder lower and closer to the ball

5. Software blocks

a. mickey.py
   Contains the main state machine.
b. movement.py
   Manages robot's movement: forward, backward, turning left and right. Managed by Pluto movement fifos: commands and completions.
   Movement requests transformed and passed to diff_driver fifo, and verified with odometry data.
c. detector.py
   Receives requests to detect a red ball in the picture, for Asus (top) camera or arm camera. Returning the exact coordinates for ball center and its radius.
   Uses Open CV algorithm based on an article. Managed by Pluto detection fifos: commands and results.
d. moveArm.py
   Controls the arm movement and gripper. Moves robot arm joints (mainly base, shoulder and elbow2) according to the (x,y) position and the radius of the ball.
   Controlled by Pluto arm fifos: commands and results.

e. grab.py
   ROS service written by Bohan Lou. Controls the fingers to grab the ball until reaching a certain amount of pressure, or to release the ball.

6. Interfaces between Pluto modules

The following ROS topics function as communication interfaces between software modules for Pluto project.

a. /pluto/detect/command
   i. Request fifo for detector.py, receives following commands:
      1. "scan_top" - Scan the image from Asus (top) camera with ball detection algorithm.
      2. "scan_arm" - Scan the image from arm camera with ball detection algorithm.

b. /pluto/detect/result
   i. Results fifo for detector.py, return DetectResult.msg:
      1. request_tag - "scan_top" or "scan_arm" from request.
      2. is_ball_detected - Either ball was detected or not.
      3. detected_x - X-axis value of the ball center.
      4. detected_y - Y-axis value of the ball center.
      5. detected_r - Radius value of the ball center.

c. /pluto/movement/command
   i. Request fifo for movement.py, receives following commands:
      1. "LEFT" - Make a single angular step left.
      2. "RIGHT" - Make a single angular step right.
      3. "FORWARD" - Make a single linear step forward.
      4. "BACKWARD" - Make a single linear step backward.

5. "STOP" - Stop the movement.
6. "FINE" - Make the motion steps smaller, when closer than 2.2 meters to the ball.

d. /pluto/movement/done
   i. Completion fifo for movement.py, always return "move_done" when last request command completed.
e. /pluto/emergency_stop
   i. Safety measurement for sudden, but graceful stop.
   ii. Values:
      1. True - Stop movement and wait for False.
      2. False - Continue algorithm flow and movement.
   iii. Default value at star is: True. Must be changed to: False, to run.
f. /pluto/move_arm/command
   i. Receives the commands:
      1. "INIT" - Lower arm to clear the view for top camera.
      2. "INIT_ARM_SCAN" - Move arm to optimal position for arm camera scan.
      3. "ARM_PICK" - Attempt to pick the ball.
g. /pluto/move_arm/done
   i. Receives completion message of:
      1. "init_arm_done" - First initialization completed.
      2. "init_arm_scan_done" - Arm scan initialization completed.
      3. "[joint]_done" - Joint movement completed.

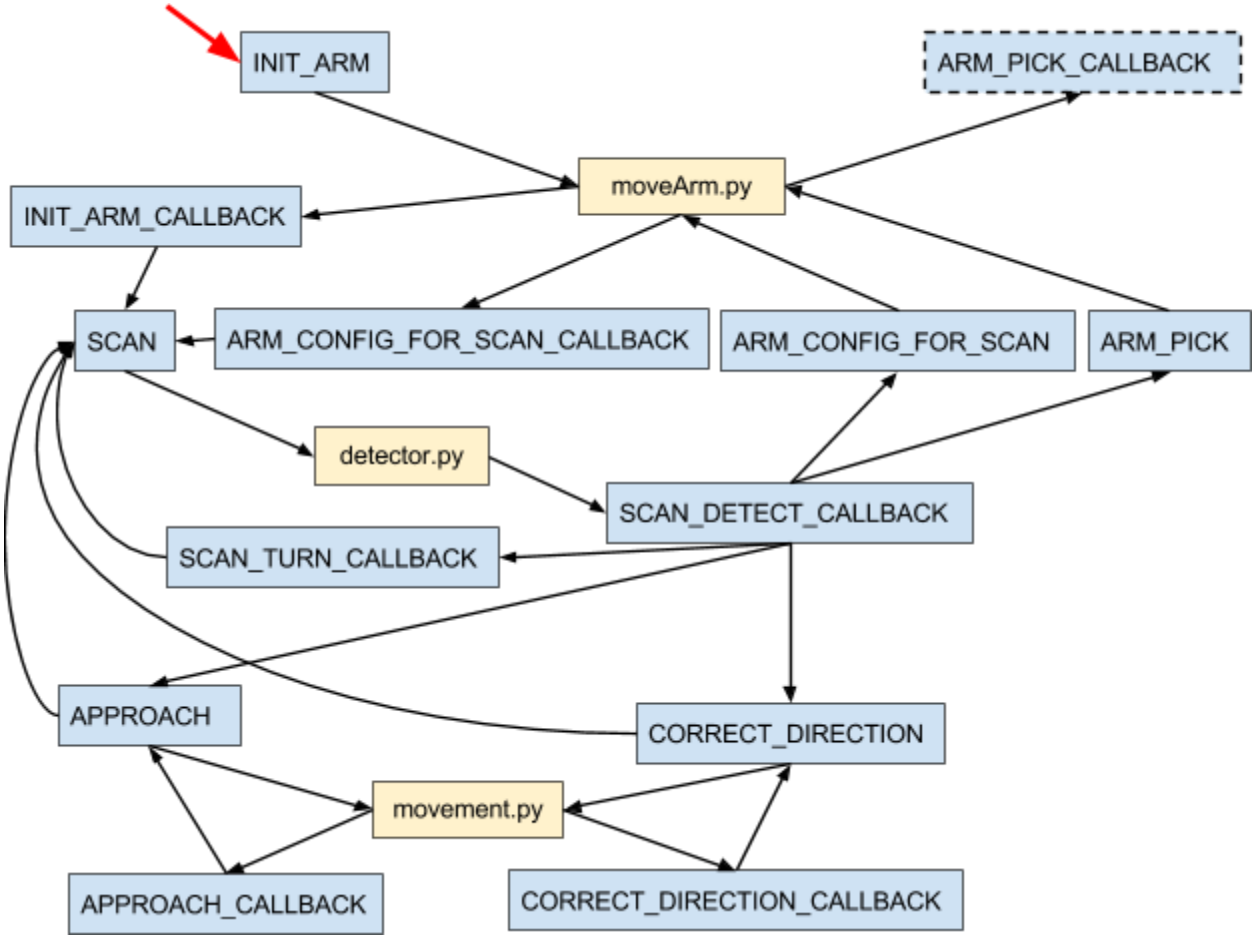
The following ROS service was used to grab the ball

a. /pluto/gripper

i. Receives command ‘grab_service("Open", -1.0)’ to open fingers
ii. Receives command ‘grab_service("Close", 0.04)’ to close fingers on object
iii. Returns float != 0 if successful

7. States diagram

Initial state is INIT_ARM.
Target state is ARM_PICK_CALLBACK.



8. Arm

The class MoveArm in “moveArm.py” controls the joints of the arm, using arm camera detection.
It receives 3 external commands from Mickey and does the following:

1. MOVE_INIT
    a. Send predefined constant values to arm joints, to lower the arm and clear the top camera view
    b. Publish completion message
2. INIT_ARM_SCAN
    a. Send predefined constant values to arm joints to move the arm in optimal position of arm camera scan
    b. Publish completion message
3. ARM_PICK
    a. Assumption: the robot is in predefined distance from the ball, the ball is seen in the arm camera
    b. Publish arm scan command to receive ball radius and (x,y) position
    c. Attempt bringing the arm closer to the ball by centering the ball in the picture. Using transition between states and arm camera request after each state - to receive an updated picture:
        i. STATE_MOVE_BASE - move base left or right according to the x position of the ball
        ii. STATE_MOVE_ELBOW - move elbow2 down or up according to y position of the ball
        iii. STATE_MOVE_SHOULDER - move shoulder down to lower the arm

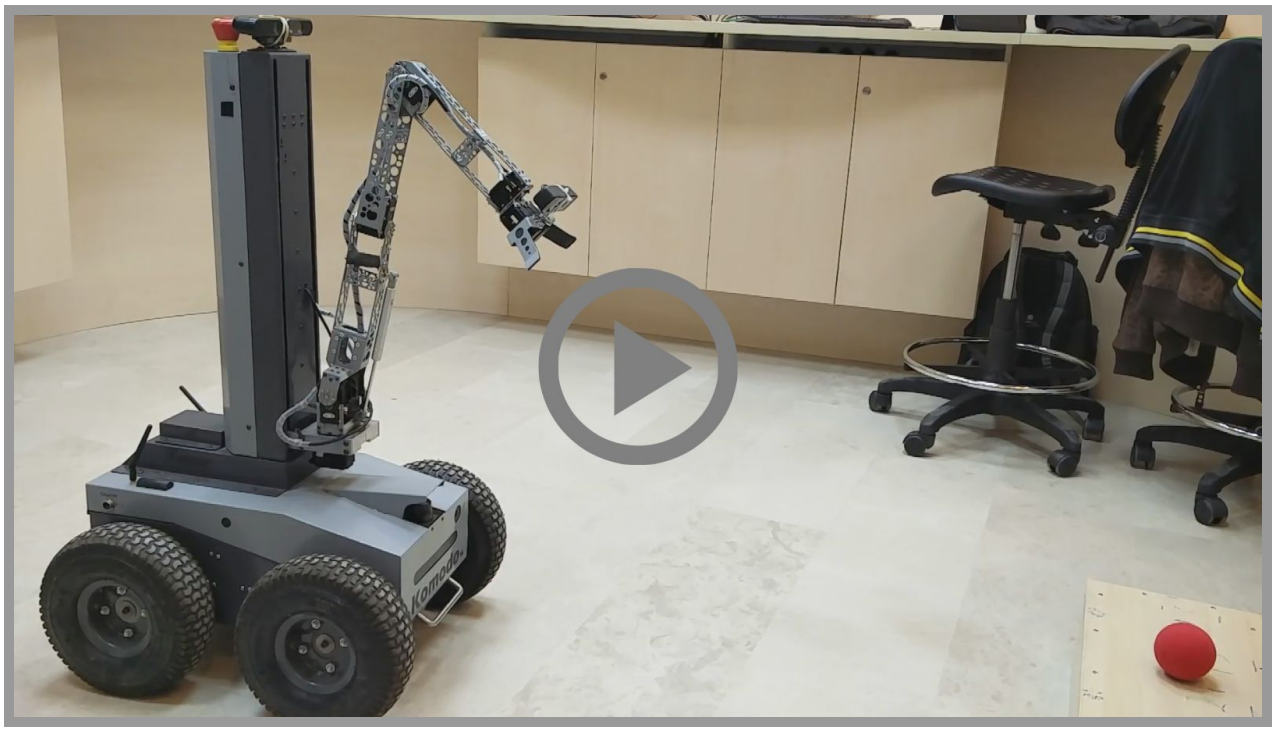
iv. STATE_PREPARE_TO_GRAB - when the ball's radius almost fills the entire picture, move the shoulder and the elbow with constant values
v. STATE_GRAB - close the fingers to grab the ball and raise the arm after grabbing the ball

9. Safety features

a. Front obstacle detection
    i. Uses front laser to detect obstacles.
    ii. In case of obstacles detected closer than 1.1 meter for top camera scan, and closer than 0.25 meter for arm scan, from the robot, will pause algorithm and any movement until obstacle is removed.
    iii. When obstacle is removed, continue to run from the same state.
b. Sides obstacle detection
    i. Uses left and right sonar sensors to detect obstacles.
    ii. In case of obstacles detected closer than 0.5 meter from robot sides, will pause algorithm and any movement until obstacle is removed.
    iii. When obstacle is removed, continue to run from the same state.
c. Emergency stop
    i. Graceful way to stop the robot with command to /pluto/emergency_stop.

10. Demonstrations

a. Demo 1



b. Demo 2

11. Git

https://github.com/a-l-e-x-d-s-9/pluto.git

12. Run instructions

a. Pull pluto git and install it into “catkin_ws/src” folder on Komodo robot.
b. For Komodo, run in terminal:
```
roslaunch pluto pluto.launch
```
c. For Gazebo simulation, run in terminal:
```
roslaunch pluto pluto_gazebo.launch
```
d. Run in terminal:
```
rostopic pub -1 /pluto/emergency_stop Bool -- False
```