MEAM 520 Final Project: Vision Based Pick-and-Place

Zheyuan Xie, Zixuan Lan

December, 2018

1 Introduction

1.1 Background

In the course we learned how to derive the forwards and inverse kinematics; we also studied multiple planning methods to generate collision-free trajectories. However, we haven't incorporate any sensing/perception part to the robot. In this project, we want to integrate computer vision techniques with the content of this course and to develop a vision-based robotic system that is capable of performing practical pick-and-place tasks which can be useful for manufacture and assembly.

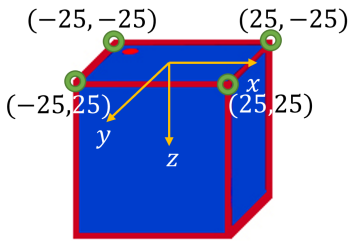
1.2 Task Definition

The objective of this project is to implement vision based pick-and-place with a 7 DOF manipulator arm equipped with a gripper and a vision sensor. We choose the KUKA iiwa 14 manipulator (Figure 1a) with a ROBOTIQ 85 gripper as our experiment platform since a similar platform is readily available in rhw GRASP Lab. However, in this project all of our works are done in a robot simulator.

The manipulator needs to grasp an object and transport it to a designated place, based only on onboard sensing, without any external localization (i.e. VICON). The environment is static and all obstacles are known to the robot. The robot should employ motion planning and obstacle avoidance technique to safely operate in the environment.

The manipulated objects are randomly spawned in the environment, and are identical in shape and color (Figure 1b). Each object is a cube with an edge length of 50mm and a mass of 125g. All faces are colored in blue, and all edges (a 2mm border of each face) are colored in red. On each of the six faces of the cube there's a red dot near one of the four corners, which indicates the orientation of the cube. In another word, we establish the object coordinate frame according to the red dot on the face the gripper will approach. The four corner of this surface is defined as object feature points M_i. Starting from the red dot M_1, the other three corners in clockwise direction are M_2, M_3, and M_4.

(a) KUKU iiwa 14 manipulator



(b) Manipulated object

Figure 1: Task objective

2 Method

In this section we are going to describe the method we use to achieve the automated pick-and-place process. The workflow of the robotic system is illustrated in Figure 2.

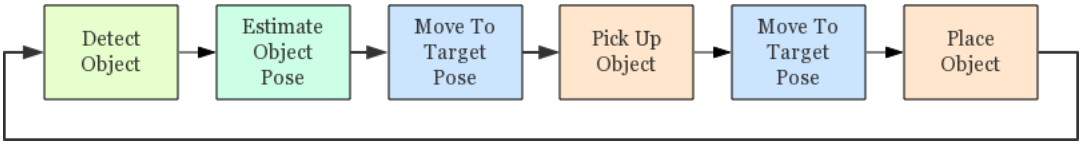


Figure 2: System Workflow Diagram

The robot first detect the object with a vision sensor, using image processing technique described in subsection 2.2. The pose of the object is estimated by solving a perspective-n-point problem (PnP), which is described in subsection 2.3. Then the robot will move to pick up the object and transport it to a designated location. The motion planning and motion planning is described in subsection 2.4. The basic kinematics of the robot is derived in subsection 2.1. Other issues and the overall task-planning will be stressed in subsection 2.5.

2.1 Manipulator Kinematics

2.1.1 Forward Kinematics

The coordinate frames of the robot are defined using the Denavit-Hartenberg, or DH convention. Table 1 lists the DH parameters, where L1,L4,L6,L8 are the length for link 1,4,8 and 8(end effector). Homogeneous transformation from Link i to Link $i-1$ is represented as a product of four basic transformations

$$
\begin{aligned}
T_i^{i-1} =& \mathrm{Rot}_{z,\theta_i}\mathrm{Trans}_{z,d_i}\mathrm{Trans}_{x,a_i}\mathrm{Rot}_{x,\alpha_i} \\
=& \begin{bmatrix} c_{\theta_i} & s_{\theta_i} & 0 & 0 \\ -s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
=& \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{1}
$$

Table 1: D-H parameters

Link	a_i	α_i	d_i	θ_i
1	0	0	L_1	0
2	0	$-\pi/2$	L_2	$\theta_1^* + \pi$
3	0	$-\pi/2$	0	$\theta_2^* - \pi$
4	0	$\pi/2$	L_4	θ_3^*
5	0	$\pi/2$	0	$\theta_4^* - \pi$
6	0	$-\pi/2$	L_6	θ_5^*
7	0	$\pi/2$	0	$\theta_6^* + \pi$
ee	0	0	L_8	θ_7^*

The transformation matrix from the end-effector to the base frame are given by

$$
T_e^0 = T_1^0 T_2^1 T_3^2 T_4^3 T_5^4 T_6^5 T_7^6 T_e^7. \tag{2}
$$

If the transformation from the object frame to the end effector (vision sensor) frame T^e_{obj} is known. We can also add the target object to our kinematic chain and get its relative pose to the robot base by

$$T^0_{\text{obj}} = T^0_e T^e_{\text{obj}}. \tag{3}$$

The method for estimating T^e_{obj} is introduced in Section 2.2 and 2.3.

2.1.2 Inverse Kinematics

After obtaining the relative pose of the object, we can infer the desired pose the gripper need to achieve before approaching and grasping the object. Getting the gripper moving to the desired pose involves solving an inverse kinematics problem. Since the KUKA iiwa 14 is a 7 DOF redundant manipulator, the solution to inverse kinematics is generically nonunique. To obtain a closed-form solution is difficult, therefore we switch to numerical methods.

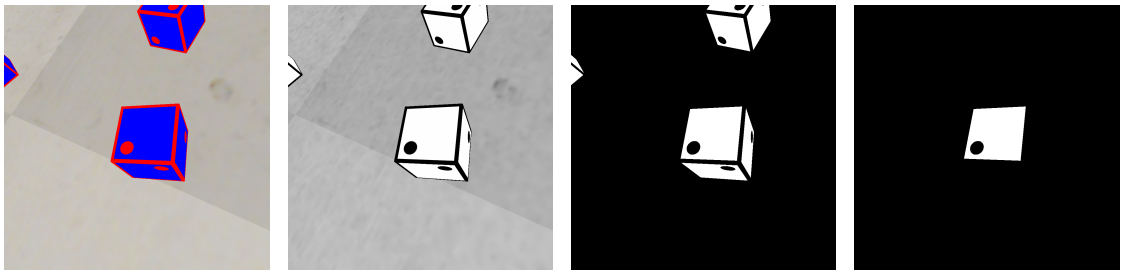
We use the `robotics.InverseKinematics` in MATLAB Robotics System Toolbox [1] to obtain the numerical solution of inverse kinematics. It uses the Broyden-Fletcher-Goldfarb-Shanno (BFGS) gradient projection algorithm to iteratively search for feasible pose around a initial configuration q_0 given tolerance. The BFGS gradient projection algorithm is a quasi-Newton method that uses the gradients of the cost function from past iterations to generate approximate second-derivative information. The algorithm uses this second-derivative information in determining the step to take in the current iteration. A gradient projection method is used to deal with boundary limits on the cost function that the joint limits of the robot model create. The direction calculated is modified so that the search direction is always valid.

2.2 Image Processing

The object feature points M_i are projected onto the image plane at m_i. To find the m_i's corresponding pixel coordinates (x_i, y_i), we need to detect the four corners of a cuboid surface.

Assuming that the target objects are colored distinctively in blue, we can apply color image segmentation to simplify the vision problem. The first step is to separate the blue (B) channel from the RGB image. The blue region of the image is extracted by applying a 0.7 threshold to the blue channel as shown in Figure 3c. If more than one faces or objects are present in the view, there will be multiple connected regions. We calculate actual number of pixels in each region and find the region with maximum area (number of pixels) as shown in Figure 3d.

Note that the maximum area region we find have a hole in it which indicates a feasible gripper orientation and serves the purpose of foolproof. The coordinate of the center of mass of this hole region is computed as a reference point $p_f(x_f, y_f)$.

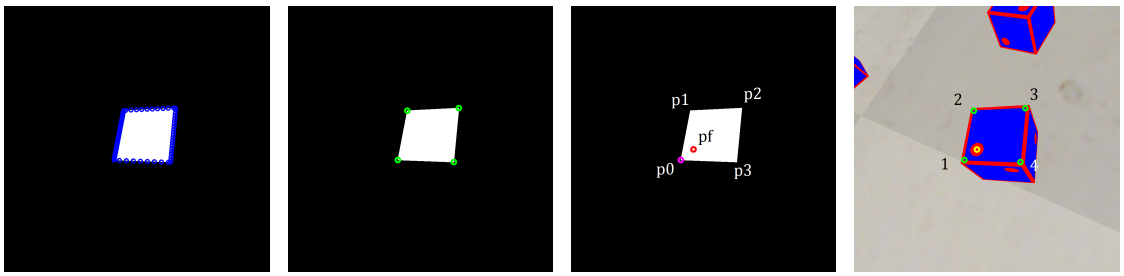


(a) Original (b) RGB Separation (c) Threshold (d) Maximum Area

Figure 3: Image Preprocessing

We perform a flood-fill operation on the maximum area region to eliminate the hole and get a solid quadrilateral shape. To find the 4 corner of this shape we use Harris corner detector[2]. The pixels are sorted in descending order according to corner metric and the first four pixels are detected as corner points. Compute the L2 distance between the reference point p_f and all four corner points. The corner point with smallest distance to p_f is assigned as the image point m_0 of the first object point M_0.

We compute the cross-product $\overrightarrow{m_0p_f} \times \overrightarrow{m_0p_i}$ $(i = 1, 2, 3)$ to determine the correspondence between the rest image points m_1, m_2, m_3 and detected corner points p_1, p_2, p_3. We will have $\overrightarrow{m_0p_f} \times \overrightarrow{m_0m_1} < 0$, $\overrightarrow{m_0p_f} \times \overrightarrow{m_0m_3} < 0$, and $\overrightarrow{m_0p_f} \times \overrightarrow{m_0m_2} \approx 0$. The result after assignment is shown in Figure 4d.



(a) Harris Corner (b) Filter by Metric (c) Foolproof Detection (d) Result

Figure 4: Corner Detection and Sequencing

2.3 Coplanar POSIT Algorithm

2.3.1 Pose from Orthography and Scaling

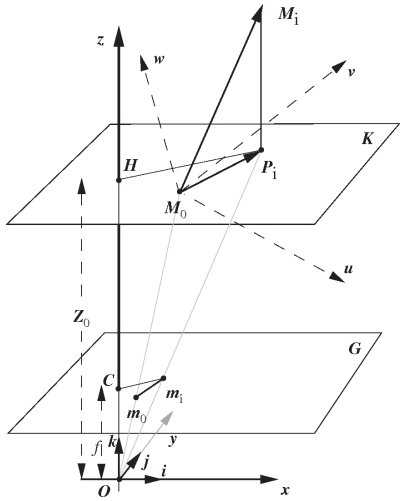


Figure 5: Perspective projection and scaled orthographic projection.

The cuboid object has feature points M_0, M_1, M_2, M_3, M_4. The images of the points M_i are m_i, and their pixel location (x_i, y_i) are determined in the previous subsection (In this section we assume the coordinates are centered at the camera principal point). We want to find the rotation matrix $\mathbf{R}$ and translation vector $\mathbf{T}$ of the object.[3]

Let $\mathbf{i}, \mathbf{j}, \mathbf{k}$ be the unit vector of the camera coordinate system expressed in the object coordinate frame. The rotation matrix can be written as

$$\mathbf{R} = \begin{bmatrix} i_u & i_v & i_w \\ j_u & j_v & j_w \\ k_u & k_v & k_w \end{bmatrix} = \begin{bmatrix} \mathbf{i}^T \\ \mathbf{j}^T \\ \mathbf{k}^T \end{bmatrix}$$

The translation vector $\mathbf{T}$ is the vector $\mathbf{OM}_0$, M_0 is the origin of the object coordinate frame. Its coordinates are X_0, Y_0, Z_0. Since the image of M_0 is known as image point m_0 at pixel (x_0, y_0), and the translation vector $\mathbf{T}$ is equal to $(Z_0/f)\mathbf{Om}_0$.

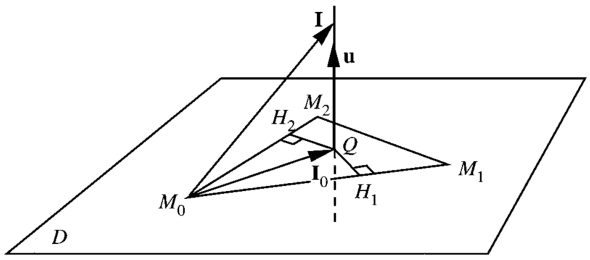


Figure 6: All vectors I whose heads project onto plane D in Q are valid solutions.

Within the iterative algorithm we need to solve the equations

$$\mathbf{M}_0\mathbf{M}_i \cdot \mathbf{I} = x_i(1+\epsilon_i) - x_0,$$
$$\mathbf{M}_0\mathbf{M}_i \cdot \mathbf{J} = y_i(1+\epsilon_i) - y_0,$$

with

$$\mathbf{I} = \frac{f}{Z_0}\mathbf{i}, \;\; \mathbf{J} = \frac{f}{Z_0}\mathbf{j},$$

and the terms ϵ_i have known values at each iteration. When solving for $\mathbf{I}$ and $\mathbf{J}$, we can construct a matrix $\mathbf{A}$ being the coordinates of the object points M_i in the object coordinate frame of the reference.

$$\mathbf{I} = \mathbf{A}^+\mathbf{x}', \;\; \mathbf{J} = \mathbf{A}^+\mathbf{y}'. \tag{4}$$

Where A^+ is the pseudo-inverse of $\mathbf{A}$ and is called object matrix.

2.3.2 Solution to Coplanar Points

In the case where the object points are coplanar, matrix $\mathbf{A}$ has rank 2 instead of 3, and the set of Eqaution 4 is determined. A geometric interpretation of $\mathbf{M}_0\mathbf{M}_i \cdot \mathbf{I} = x'$ is that if the tail of $\mathbf{I}$ is taken to be $\mathbf{M}_0$ and the head of $\mathbf{I}$ projects on $\mathbf{M}_0\mathbf{M}_i$ at H_{x_i}, the head of $\mathbf{I}$ belongs to the plane perpendicular to $\mathbf{M}_0\mathbf{M}_i$ at H_{x_i}. If the all object feature points belong to the same plane D, then the vectors $\mathbf{M}_0\mathbf{M}_i$ are coplanar and the planes perpendicular to them at H_{x_i} all intersect at a single line or at close parallel lines that are prependicular to plane D.

Solving Equation 4 for coplanar object points will wive us $\mathbf{I}_0 = [I_u, I_v, 0]^T$ and $\mathbf{J}_0 = [J_u, J_v, 0]^T$. The I_w and J_w component of $\mathbf{I}_0$ and $\mathbf{J}_0$ are 0 since the solution vector is in plane D. According to [3], we are going find a vector $\mathbf{u}$ which is perpendicular to plane D and the solution can be written as

$$\mathbf{I} = \mathbf{I}_0 + I_w\mathbf{u}, \;\; \mathbf{J} = \mathbf{J}_0 + J_w\mathbf{u}, \tag{5}$$

where I_w and J_w can be solved by use the additional fact that $\mathbf{I}$ and $\mathbf{J}$ must be perpendicular and be of the same length. This will yield two solutions for $\mathbf{I}$ and $\mathbf{J}$.

However, during implementation we found out that the solution for I_w and J_w can converge to wrong value when $\mathbf{I}_0$ and $\mathbf{J}_0$ are not given accurately. **We added an additional constraint that the camera frame z_e axis and object frame z_o axis are parallel when the robot is observing the environment.** Which means if we are approaching the object from above, then we turn the vision sensor straight down to observe the object. In that case we know the I_w and J_w are 0, the solution can therefore be written as

$$\mathbf{I} = \mathbf{I}_0 + [0, 0, \lambda_1]^T, \;\; \mathbf{J} = \mathbf{J}_0 + [0, 0, \lambda_2]^T \tag{6}$$

where λ_1 and λ_2 ($|\lambda_1| = |\lambda_2|$) are computed from $\sqrt{I_u * J_u + I_v * J_v}$ to make sure $\mathbf{I} \cdot \mathbf{J} = 0$ satisfies. Otherwise we will not get a valid rotation matrix.

We implemented the following algorithm. If the origin of the object coordinate system is not one of the image points of the image. We use an alternative approach described in [4].

Algorithm 1 Coplanar POSIT with Orientation Constraint

Input: object points M_i, image points m_i, focal length f, principal point (x_0, y_0)
Output: estimated object orientation $\mathbf{R}$ and translation $\mathbf{T}$

Construct matrix $\mathbf{A}$ from the coordinates of the object points
Compute the object matrix $\mathbf{B} = \mathbf{A}^+$
$\epsilon_{i(0)} = 0$, $n = 1$
while not converge **do**
    Solve for $\mathbf{I_0}, \mathbf{J_0}$ using Equation 4
    Compute I, $\mathbf{J}$ from $\mathbf{I}_0$, $\mathbf{J}_0$ using Equation 6
    Compute $\mathbf{i}$, $\mathbf{j}$, and Z_0 by normalizing I, J
    Compute $\epsilon_{i(n)} = (1/Z_0)\mathbf{M}_0\mathbf{M}_i \cdot \mathbf{k}$, with $\mathbf{k} = \mathbf{i} \times \mathbf{j}$
    $n = n + 1$
Compute $X_0 = x_0 Z_0/f$, $Y_0 = y_0 Z_0/f$
$\mathbf{R} = [\mathbf{i}, \mathbf{j}, \mathbf{k}]^T$, $\mathbf{T} = [X_0, Y_0, Z_0]^T$

2.4 Probabilistic Road Map Method

2.4.1 Build a PRM

We uniformly sample the configuration space at random, and discard all the samples that causes a collision. The valid sample forms our free C-space $\mathcal{Q}_{\text{free}}$. We will use this naive technique without considering the *narrow passage problem* since we are not going to encounter such situations in this lab.

We will build a PRM on the $\mathcal{Q}_{\text{free}}$. Given a set of nodes that correspond to configurations, we connect each node q to it's k nearest neighbours $q_i', (i = 1, 2, \cdots, k)$ with a weight of $d(q, q_i')$. The distance metric we use is L_∞ norm, which physically means to minimize the max joint angle a robot turns from q to q'. A MATLAB graph object is used to store this data structure.

Algorithm 2 Construct probabilistic roadmap

V = UniformSample(n), $E = \{\}$
$G = (V, E)$
for each $q \in V$ **do**
    for each $q' \in N_k(q)$ **do**
        Linearly interpolate between q and q'
        if NOT CheckCollision(q_i) for all interpolated q_i **then**
            $E = E \cup (q, q')$
Simplify G by removing duplicated edges

To check if a specific configuration q of the robot collide with the obstacle, we represent the robot by a series of line segments. Use the `detectcollision` function we are able to determine whether a line segment intersect with a rectangular object. The robot is collision-free only if all of the line segments are not colliding with the obstacles. Taking the robot geometry into account, we inflated the obstacles with a 20cm radius. We implemented an collision check algorithm as listed below in Algorithm 3.

2.4.2 Find the Shortest Path

To find a path given q_{start} and q_{goal} based on the PRM we take the following steps:

1. Find q_a and q_b, which are nodes in the PRM that are closest in C-space to q_{start} and q_{goal} configuration;
2. Search the PRM for a path from q_a to q_b with Dijkstra algorithm, and we denote the waypoints in the path by $\{q_{start}, q_a, q_1, q_2, \cdots, q_{p-1}, q_p, q_b, q_{goal}\}$;
3. Linearly interpolate between the waypoints to get the full trajectory.

Algorithm 3 Collision check for a specific configuration

```
procedure CheckCollision(configuration:q, obstacleMap:M)
    jointPositions = calculateFK(q)
    Construct line segment L_i for each link i from jointPositions
    isCollided = false
    for each object O_i in M do
        for each link L_i do
            inflate O_i by Link i's maximum cross-section radius R
            isLinkCollided = line segment of L_i collide with O_i
            isCollided = isCollided or isLinkCollided
    return isCollided
```

2.5 Task Planning

In a pick-and-place task, the robot first moves to an observe configuration $q_{\text{ob}} = [0,\ \pi/6,\ 0,\ -\pi/3,\ 0,\ \pi/2,\ 0]$. At the observe configuration, the robot detect the object and estimate its transform T^0_{obj}. Then the robot targets its end-effector pose to $T^0_{\text{obj}} \cdot 0.2\mathbf{u}$ ($\mathbf{u}$ is the surface normal vector of the target object), which is 20cm away from the surface of the object. Then the robot will plan and execute a collision path to the pose and perform a secondary observation from there and re-estimate the pose T^0_{obj}. This time the end-effector will approach to $T^0_{\text{obj}} \cdot 0.02\mathbf{u}$ which is very close to the surface and execute the grasp.

After grasping the object, the end-effector will move back a little bit along $\mathbf{u}$ and then move to the designated pose using the PRM planner. The robot returns to the observation configuration and gets ready for the next pick-and-place routine after releasing the object.

3 Evaluation

Testing on a real robot can be costly, therefore we used the virtual robot experiment platform (V-REP) by Coppelia Robotics [5] to conduct our experiment. In this section, we are going to first setup the simulation environment, then evaluate our pose estimation method and motion planning method, and finally integrate everything to perform pick-and-place tasks.

3.1 Experiment Setup

3.1.1 V-REP Scene

An experiment scene and its cooresponing obstacle map is constructed as shown in Figure 7. A KUKA iiwa 14 manipulator that comes within the integrated model library is placed at the center of the scene. A ROBOTIQ 85 gripper is attached to the end of the manipulator. A vision sensor is attached to the gripper. We let the sensor coordinate frame coincide with the end-effector (gripper) frame to simplify calculation. A table, a shelf and a bowl is also added to the scene which is used in different pick-place tasks.

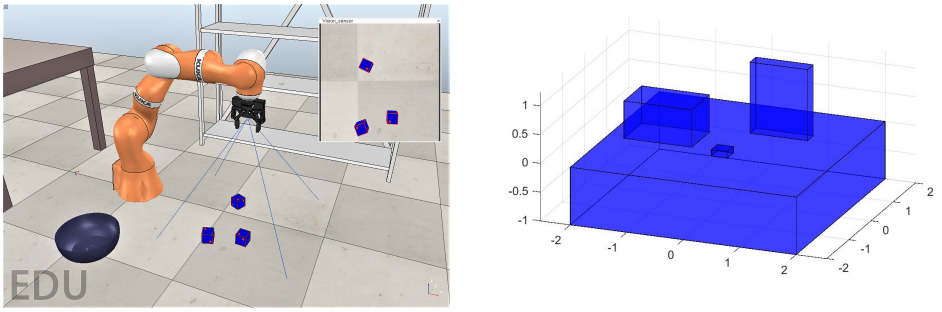


Figure 7: Experiment scene setup in V-REP

3.1.2 MATLAB Code

V-REP offers a remote API allowing to control a simulation (or the simulator itself) from an external application. The remote API on the client side is available for many different programming languages including MATLAB. We use the MATLAB remote API to control the scene we constructed. We also implemented a collection of MATLAB functions and script, including the algorithms described in Section 2, as listed in Tabele 2.

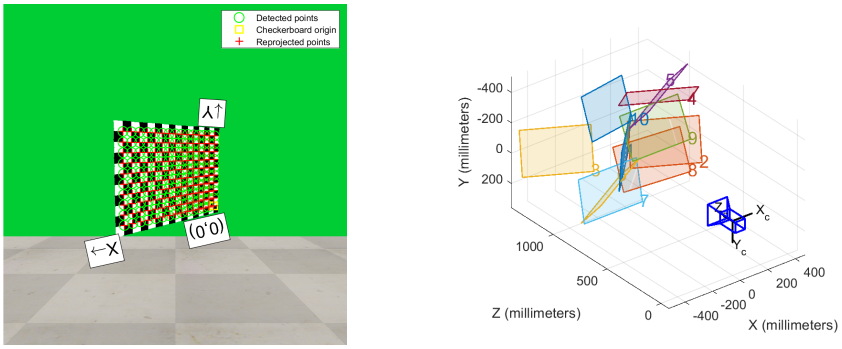
Filename	Functionality
kuka_start.m	Initialize and tart simulation.
kuka_stop.m	Stop simulation.
kuka_fk.m	Calculate forward kinematics.
kuka_getImage.m	Get sensor image from V-REP.
kuka_getCurrentConfig.m	Get current configuration from V-REP.
kuka_servo.m	Set target joint positions in v-REP.
kuka_setGripper.m	Set gripper command in V-REP.
getConfig.m	Convert q array to MATLAB struct.
getObjPose.m	Get object pose from V-REP by name.
coplanarPosit.m	Coplanar POSIT algorithm.
estimateTransform.m	Process image and estimate transform.
checkCollision.m	Check collision given configuration.
buildPrm.m	Build a PRM from obstacle map.
planPath.m	Plan a collision-free path based from PRM.
kuka_move2config.m	Move to a configuration.
kuka_move2pose.m	Move to a pose.
kuka_prmmove2config.m	Move to a configuration with PRM planner.
kuka_prmmove2pose.m	Move to a pose with PRM planner.
pickPlace.m	Pick-and-place task wrapper.

Table 2: Code implemented in MATLAB.

3.1.3 Camera Calibration

The vision sensor we used for our experiment is a perspective projection-type with an angle of view (AOV) of 60 degree and generates 1024×1024 images. The camera is an ideal pin hole camera which has its principal point located at the center of the image and there's no skew. However, we conducted experiment to find out the focal length in pixels since we do not know the distance and size of the image plane.

We used the MATLAB camera calibrator toolbox [6] to perform camera calibration. A checkerboard is created in a V-REP scene in front of the image as shown in Figure 8a. A total number of 10 images is collected by moving the checkerboard around.



(a) Checkerboard (b) Camera centric

Figure 8: Using the MATLAB camera calibrator

The calibration result gave us the camera intrinsic matrix:

$$K = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 887.05 & 0 & 512.01 \\ 0 & 886.92 & 512.59 \\ 0 & 0 & 1 \end{bmatrix}$$

3.2 Pose Estimation

To evaluate our pose estimation method, we set the robot to observe configuration and fixed the pose of the vision sensor. Then we randomly spawn the object on the ground plane in a 2m×2m area. We compare the estimated relative pose with true relative pose which is obtained by calling V-REP API `simxGetObjectPosition` and `simxGetObjectOrientation`. The process is repeated 1000 times.

Since we add the constraint that the camera z-axis is parallel to the object z-axis, the estimated object pose relative to the camera frame has the form

$$T^e_{\text{obj}} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & t_x \\ \sin\theta & \cos\theta & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

We use a quiver plot to demonstrate position error. Each small arrows is pointing from true position to estimated position, as shown in Figure 9. The maximum position error is 0.0203m (2cm). However, the maximum position error in X or Y direction is 0.0150m. Since the distance between the gripper jaw is 85mm larger than the object width (50mm), a 15mm estimation error in one direction is tolerable.

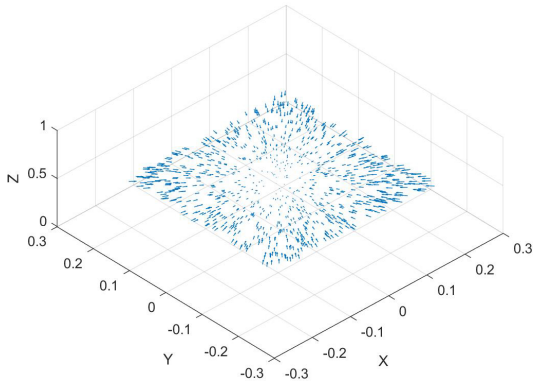


Figure 9: Position estimation error.

The estimation error in angle θ is demonstrated in Figure 10. The maximum angular error is less then 0.015 rad (0.86 deg), which is accurate enough for our pick-and-place task.

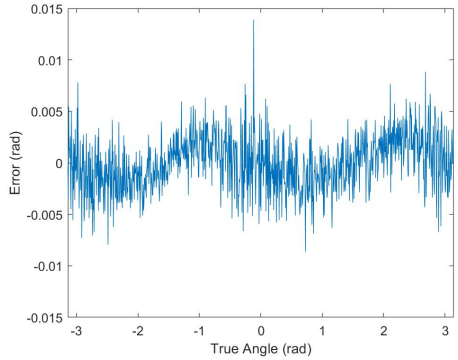
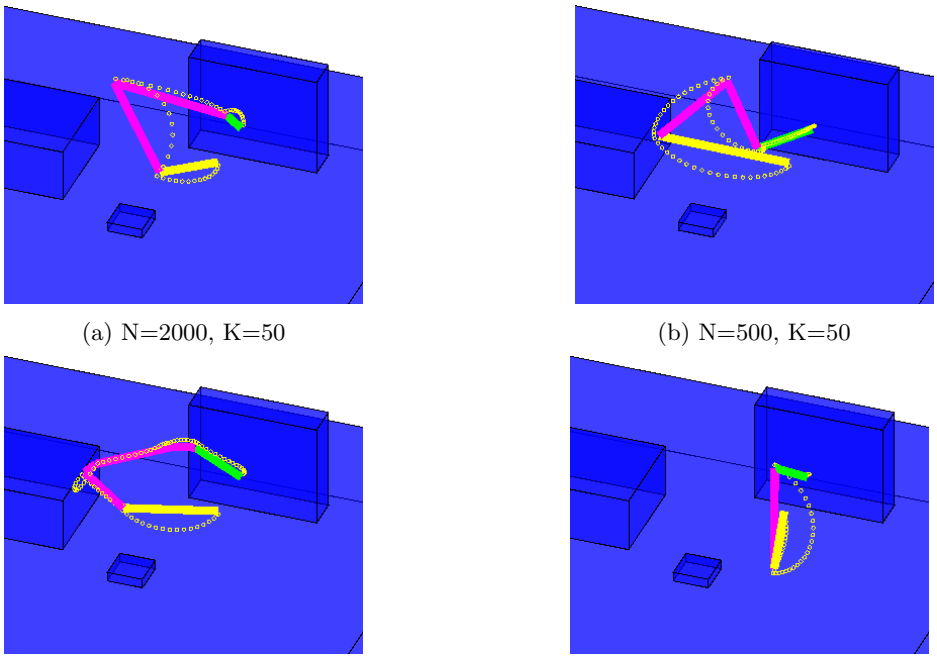


Figure 10: Angle estimation error.

3.3 Path Planning

In our path planning algorithm, two major parameters that affect the performance are number of nodes N and number of neighbours K that each node connects to. To evaluate the effect of the two parameters, we constructed four probabilistic roadmaps, and plan a path from $q_{\text{start}} = q_{\text{observe}}$ to $q_{\text{goal}} = \text{IK}(T_{\text{target}})$ using each of the roadmaps. The PRM building time and total joint angle turned along the planned path are recorded. The results are reported in Table 3. Figure 11 shows the planned path using each of the roadmaps, the green segment connects q_{start} to q_a, and the yellow segment connects q_b to q_{goal}.



(a) N=2000, K=50

(b) N=500, K=50

(c) N=500, K=10

(d) N=100, K=10

Figure 11: Path planning result with different N and K.

Table 3: PRM with different graph complexity

Trial	N	K	Computation Time	Total Angle Turned
1	2000	50	2475.74 s	15.22 rad
2	500	50	652.85 s	25.37 rad
3	500	10	161.00 s	27.40 rad
4	100	10	30.36 s	15.73 rad

From the first three trials we can observe that increasing N and K will increase the performance at the cost of computation time. However, the fourth trail with $N = 100, K = 10$ is an exception since there are too few nodes. In cases like the fourth trail, though the total joint angle turned is smaller, the path is less safe and not guaranteed to be collision-free.

3.4 Task Integration

To integrate everything, we performed the pick-and-place procedure described in Section 2.5 in six consecutive runs, the system succeeded to place the object to a designated position in every run. Here is a video demo for our pick-and-place system: `https://youtu.be/R8ArL_oiKQU`.

Table 4 shows the mean execution times of the different parts of the presented approach, although computational and execution speed was not a focus of this project. The main fraction of time is spent for

path execution (i.e. move from and to observation configuration, pick-up pose, and target pose). We did not take the time spent for building PRM into account since it only needs to execute once if the environment does not change.

Table 4: Mean execution times.

Task	Mean time (s)	σ time (s)	Fraction (%)
Image Processing	0.476	0.0199	2.0
Pose Estimation	0.552	0.0337	2.3
Path Planning	2.614	0.1266	11.0
Path Execution	12.71	1.590	53.5
Object Manipulation	7.40	0.2502	31.2
Total	23.752	-	100

4 Analysis

4.1 Limitations

The vision-based object detection algorithm works well in the simulation environment, and the pose estimation algorithm achieved satisfactory accuracy in the pick-and-place task. However, the practical use on a physical robot of our method remains untested. In real world scenarios, noise, camera distortion, change in illumination, and other factors can affect the robustness and accuracy of the system. If we want to extend our work to an real robot, these are all important factors to be considered.

The system error in pose estimation is clearly seen in Figure 9. The error grows as the object moves away from the optical axis of the camera. It may be caused by camera calibration problem. Also, we find the coplanar POSIT algorithm in [3] does not work if we follow the solution in Equation 5. We modified the algorithm and restricted the use to only certain occasions when camera z-axis is parallel with object z-axis. Further investigation is needed into this subject if we want to apply our method in more complex situations.

The PRM planner is not finding the optimal path, and sometimes performs bad. This is caused by insufficient sampling. The KUKA iiwa 14 robot is a 7 DOF robot with 7 joint parameters. If each joint parameter is discretized into 10 values, it would require 10^7 nodes. However, we only sampled 2000 nodes. Building a PRM with 10^7 nodes requires a huge amount of time, therefore, we think this method is not good for robot that have high DOF due to the curse of dimensionality.

4.2 Summary

In this project, we analyzed the kinematics of a KUKA iiwa 14 manipulator, implemented object detection, pose estimation, and path planning algorithm, and achieved vision-based pick-and-place in V-rep robotic simulator.

References

[1] Inc. The MathWorks. *Robotics Systems Toolbox*. 2015–2018.

[2] Chris Harris and Mike Stephens. "A combined corner and edge detector." In: *Alvey vision conference*. Vol. 15. 50. Citeseer. 1988, pp. 10–5244.

[3] Denis Oberkampf, Daniel F DeMenthon, and Larry S Davis. "Iterative pose estimation using coplanar feature points". In: *Computer Vision and Image Understanding* 63.3 (1996), pp. 495–511.

[4] Philip David et al. "SoftPOSIT: Simultaneous pose and correspondence determination". In: *International Journal of Computer Vision* 59.3 (2004), pp. 259–284.

[5] M. Freese E. Rohmer S. P. N. Singh. "V-REP: a Versatile and Scalable Robot Simulation Framework". In: *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*. 2013.

[6] Jean-Yves Bouguet. "Matlab camera calibration toolbox". In: *Caltech Technical Report* (2000).