6 DoF Visual Pose Estimation with Occlusion Handling in Simulated Space Environment

Zheng Xu

CMU-CS-21-130
August 2021

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Howie Choset, Chair
Matthew J. Travers

Submitted in partial fulfillment of the requirements for the Fifth Year Master's Program.

Keywords: Visual Pose Estimation, Computer Vision, Image Segmentation, Extended Kalman Filter

Abstract

Peg-in-hole insertion is a task where the robot attempts to position the end-effector at a given socket. Conventional setups have a camera aligned with the end-effector to get an unobstructed view of the socket. In our specific setup for space satellite service, the Mission Robotic Vehicle (MRV) attempts to insert a payload placed on a robot arm into the socket on the client satellite with visual feedback from a camera placed on the base satellite. Due to the nature of this setup, the robot arm can block some part of the visual feedback and the socket may not be directly visible. In this work, we develop a simulated environment that best reproduces the real world environment and propose two methods of dealing with occlusion in this scenario based on computation resources. The first method uses knowledge of the pose of the robot arm to estimate occluded areas. The projection of the arm onto the image plane forms a mask of occluded regions and the tracker will avoid masked regions. The second method is similar to the previous method except the mask is generated from learning-based image segmentation. Both methods are robust with various levels of occlusion. The first method requires very low computational power. The second method can function without knowledge of arm specifications and poses and provides a better estimate of occlusion. Our observation shows that both methods improve tracking accuracy and reduce the rate at which the tracker loses track due to occlusion.

Acknowledgments

I would like to thank my advisor Professor Howie Choset for his guidance and help during my years at CMU. He has given me direction in my research and provided great feedback for my work. I would also like to thank my group members in the NG Space Arm group that helped me in this project: Ian Abraham, Vitaliy Fedonyuk, Daniel Vedova, Kirtan Patel, Raghavv Goel, Alexander Bouman and Anna Yu. They have provided amazing help in setting up the simulation environment, debugging the code, providing insight to issues and reviewing my work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Background

Currently, there are around 6500 satellites in orbit. Around 3100 are inactive and many of the remaining are reaching end of lifespan without additional support. Deorbiting these satellites is both ineffective and dangerous. The alternative would be an in-orbit satellite-servicing mission that can extend the lifespan of satellites by delivering life extension payloads. However, current satellite service and repair, operated by space walking astronauts, is costly and risky. This has led to companies such as the Northrop Grumman Corporation (NGC) to develop solutions to automate the service and repair with robots. NGC's solution involves three components, like whats shown in Figure 1.2. The Mission Robotic Vehicle (MRV) deliver Mission Extension Payloads (MEPs)to client satellites in low-earth orbit to provide necessary fuel and electricity to prolong the lifespan of the client satellite. The arm needs to maneuver the MEP towards the client satellite and finish a docking procedure. Before the docking procedure begins, the MRV will adjust its velocity and position such that the client vehicle is within camera view, with in robot arm's reach and relatively stationary to the MRV. During the docking procedure, a cylindrical contact rod needs to be inserted into the socket on the client satellite without forceful contact that can perturb the motion of the client satellite or damage the MEP.

In order to maneuver the MEP to the client satellite and dock it precisely, the robot needs "position sensor" to obtain accurate pose estimate of the satellite. In this case, the "position sensor" is a camera mounted on the base of the satellite. The camera will provide 6 DoF pose estimation of the client satellite for the robot arm controller to accomplish docking.

1.1 Visual Tracking and Pose Estimation

Visual tracking and pose estimation is a computer vision task where the algorithm takes a camera feed of an object (image or image sequence) as input and outputs the 6 DoF pose of the object. The algorithm has knowledge of the shape of the object (generally in the form of 3D model) and the parameters of the camera. The algorithm can also take in additional input like depth information. In our case, the pose estimation algorithm has access to camera feed and joint position of the robot arm.

Visual tracking and pose estimation can be optimization based or learning based. Optimization based methods extract information in an image or a sequence of images with conventional

Figure 1.1: Real Robotic Arm in Space

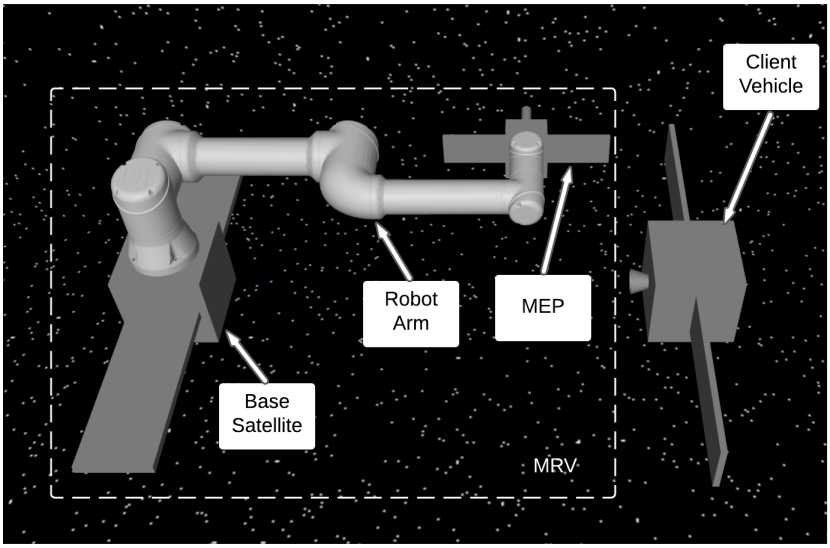


Figure 1.2: Docking setup in space, the robot arm is mounted on top of the base satellite

methods such as feature extractor, optical flow or homography matrix, and use feature matching to calculate a pose that better suits the object to the information. Learning based methods [11] use a neural network with image or sequence of images as input and directly outputs pose of the object. There are also attempts at using a hybrid of these two [21], where the initial output of a neural network is further optimized by feature matching. We will do in depth analysis and comparison with our work in Related Works Chapter.

1.2 Challenges

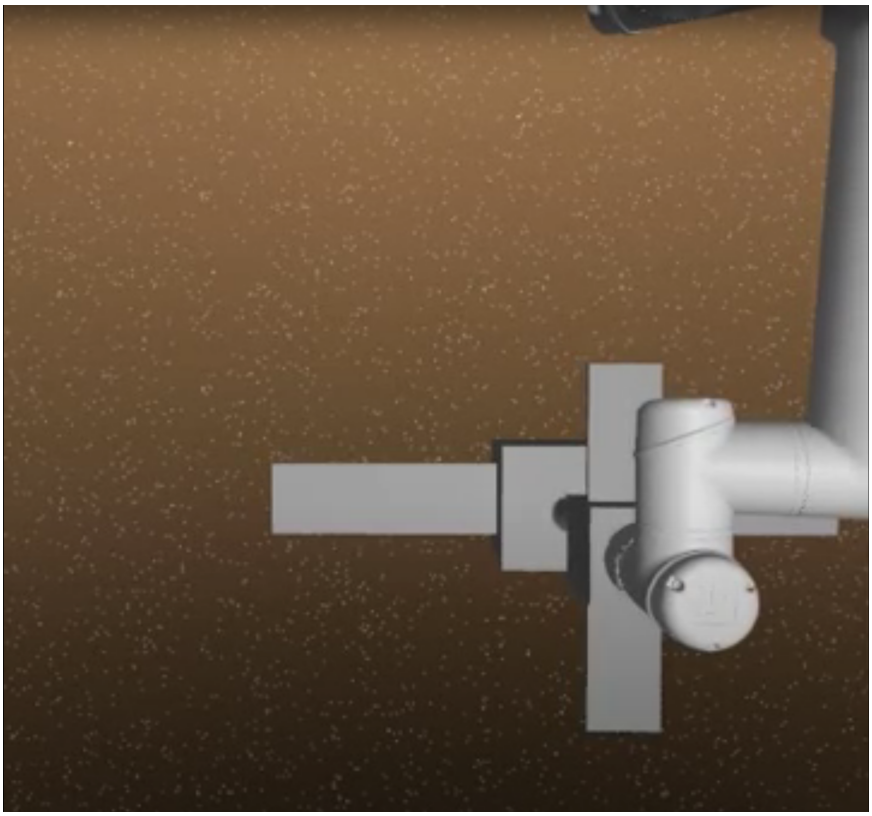


Figure 1.3: Occlusion Example from Camera Input. The socket is not entirely visible, and part of the client satellite is blocked.

The major challenge in the visual position estimation is occlusion. In this specific hardware setting, we do not have a camera positioned on the MEP, neither do we have one on the joints of the arm. From the camera inputs on the base of the MRV Figure 1.3, the socket is unavoidably blocked by the MEP during the docking procedure. This occlusion means that in order to approach the socket and successfully dock, we have to rely on retrieving the pose of the client satellite as a whole and calculate the position of the socket from rigid transform. Further more, the arm will also block parts of the satellite, which makes tracking the satellite unstable. At worst cases, the tracker will lose track of the client vehicle, failing the entire docking process.

The second challenge in this project is the scarcity of computation power. Some works in visual tracking opt for elaborate deep neural network structures [7] [12] or structure from motion (SfM) [20] for visual pose estimation. Running such algorithms at our requirement of frequency and resolution requires GPU support, which is not available in our hardware requirement. Apart from lack of GPU support, the computational units on the satellite of Northrop Grumman Corporation (NGC) have to handle the computation of many tasks, and computational resource available for visual pose estimation is scarce. To run the pose estimation at frequency suitable for arm control (2-5Hz), the algorithm has to be computationally cheap.

The third challenge is the robustness of tracking. Many learning based visual tracking uses end-to-end deep network structure to do visual tracking. In their setup, the input to the neural network is the camera input and the output is the 6 DoF pose directly. While their method yields acceptable accuracy and success rate, there are no indication of tracking failure. Because all the computation happens within the neural network the actual accuracy of tracker during run time is obscure from the user. In the context of this project, this kind of behavior would cause erroneous control in the arm that can damage the arm, the MEP or the client vehicle. Our tracker needs to have a clear indication of tracker accuracy and report failure when it occurs.

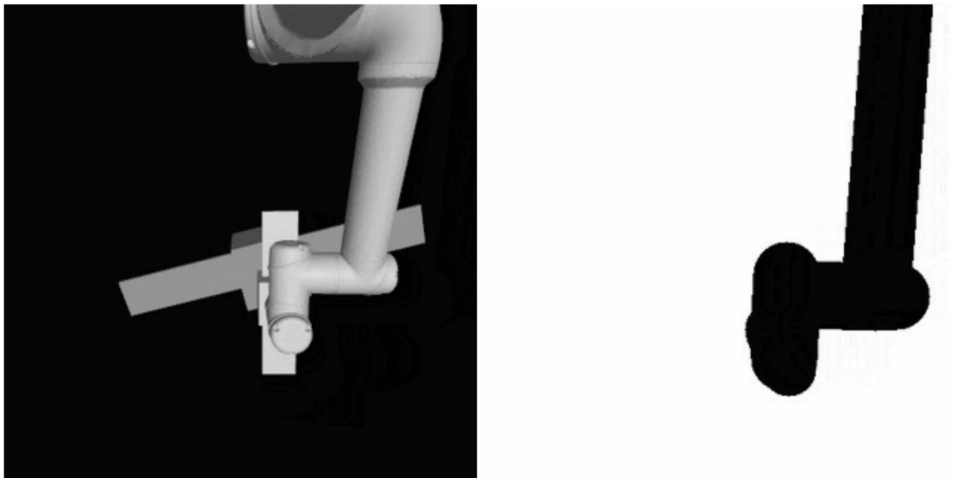


Figure 1.4: Illustration of occlusion mask. On the left is the original image and on the right is the occlusion mask.

1.3 Contribution

This works primary contribution is a 6-DoF vision tracking pipeline Figure 1.5 that provides accurate tracking that is minimizes the inaccuracy caused by occlusion and requires minimal computational power. The occlusion detection uses forward kinematics to create a mask of occluded region, like in Figure 1.4. The visual tracker will use this mask as additional input and avoid detecting feature points within the occluded region. The output of the tracker is passed through a Hybrid Extended Kalman Filter that switches dynamic models based on force sensors. The Hybrid EKF result is also used to reinitialize the tracker when there are signs of losing track. The entire system runs at over 60Hz on a conventional desktop and can run at (estimated) 4-5Hz on NGC hardware that does not have much computational resource. In Chapter 2 we will describe in detail the setup of the environment we test in. In Chapter 3 we develop a Hybrid EKF and its integration into the system. In Chapter 4 and 5 we present the occlusion detection and handling techniques using joint position and learning-based image segmentation.

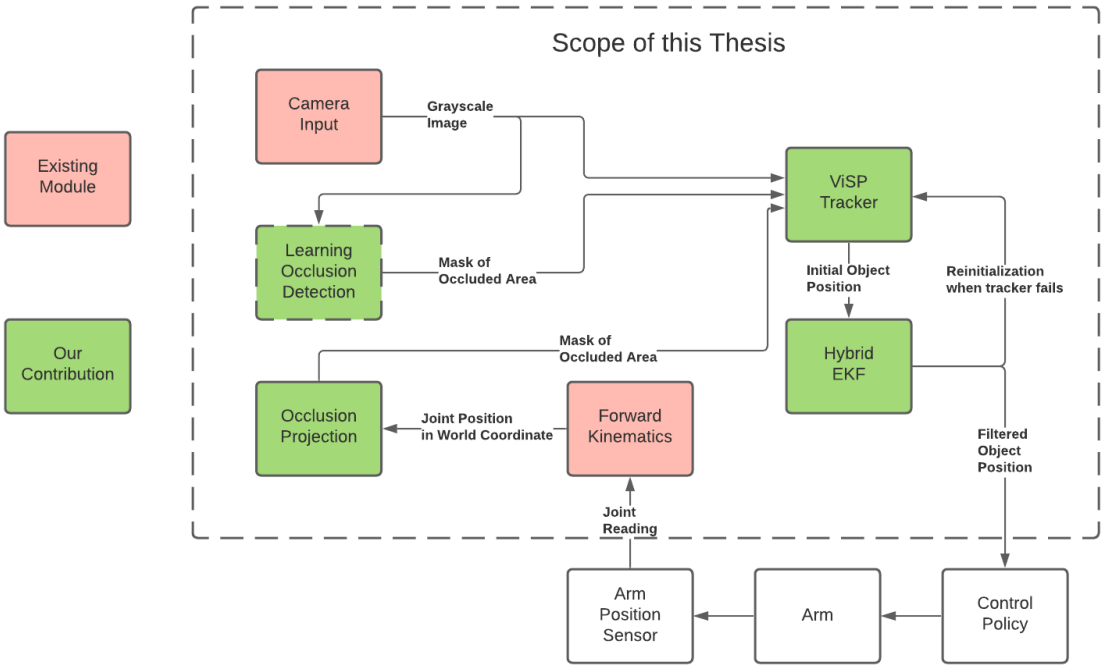


Figure 1.5: Pipeline of visual tracking and pose estimation. Red modules are existing modules and green modules are our contributions.

Chapter 2

Development Environment

Because the experiment needs to mimic the actual operation environment in space, setting up a testing environment in real world would be difficult. For instance, it would be almost impossible to create a weightless environment with free floating base and client satellite movement, and the lighting condition in space would require extensive setup to recreate. Instead, the entire experimental environment is set in simulation.

2.1 MuJoCo Simulation

Multi-joint dynamics with Contact (MuJoCo) [2] is a physics engine that facilitates fast and accurate simulation in robotics. It specializes in model-based-optimization and optimization through contacts. It also includes cross-platform rendering engine that facilitates simulation of camera inputs.
The reason we choose MuJoCo is because it is the fastest and most accurate simulator among related packages (Gazebo, PyBullet [10]), and does not require back-end support like ROS, which can increase complexity of development and deploy. The API for MuJoCo in Python is extremely simple. Most changes in the simulation environment can be changed with one line commands or included in XML configuration files.

2.1.1 Setup

Figure 1.2 demonstrates the experimental setup of the scenario. Both the base and client satellite are free floating in weightless environment. There is a 6 DoF UR5e arm fixed on top of the base satellite. A camera is placed in the front of the base satellite. The camera has a 139 degree FOV on x and y axis and outputs a 400 by 400 grayscale image. The reason for choosing this format and resolution is to reduce input size and increase frequency of the tracker.

The client satellite will have configurations wrt the MRV base listed in Table 2.1. The test case is limited to this space because the camera needs to capture the majority of the client vehicle in the image. Outside of this region the camera cannot see the client vehicle and tracking cannot proceed.

The tracking algorithm has information on the following parameter and data to conduct tracking.

- Camera matrix P
- Camera image input I
- Arm geometry G_a
- Joint configuration $\mathbf{q} = \{q_j, j \in 1...6\}$
- Client satellite geometry G_c.

Field Name	**Range**
x/y pos	[-0.15, 0.15]
z position	[0.8, 1.0]
roll/pitch/yaw	$[-15°, 15°]$

Table 2.1: The table shows the initial condition of client vehicle

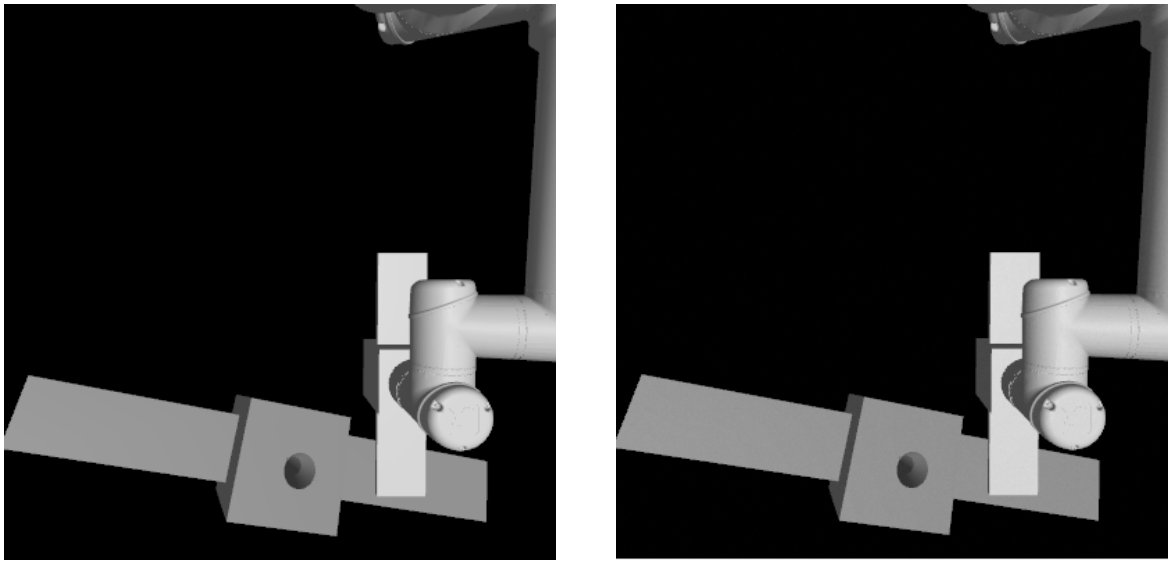
2.1.2 Image Noise and Lighting

The first step of recreating realistic camera input is to recreate similar lighting condition in space. The background is simulated with a black skybox with no illumination. The illumination on the satellite is set to in regards to adding sensor noise into the camera input, similar efforts have been done in simulating real camera input [11]. In our case our image is in grayscale format, so we are adding a Gaussian distribution $\mathbf{N}(0, \sigma)$ with $\sigma = 0, 1, 2$ to each pixel value. The result of no image noise and $\sigma = 2$ image noise are shown below. Note that while at this size the noise is hardly discernible by human eyes, it can influence the performance of visual pose estimation and tracking.

2.2 ViSP

The baseline tracker that we are building upon is called Visual Servoing Platform (ViSP) [4] [18]. It is a modular cross platform library that allows prototyping and developing applications using visual tracking and visual servoing techniques. The module that we are using in this project is model-based tracker [5]. The model-based tracker works as in Figure 2.2 (Note that mask input is not in the original model-based tracker. We made modifications to the tracker to accept a mask input). It uses a hybrid of moving-edge tracker and KLT tracker to track object with pronounced edges. The tracker performs well under perturbation like image noise. However, the tracker is not reliable with occlusion.

The problem with default ViSP tracker is that it extracts features regardless of whether it belongs to the object itself. In the first example, the tracker got the majority of features on the client satellite. However, the tracker also picked up features from the robot arm that deviated the measurement from the ground truth. In the second one, the tracker is completely disturbed and didn't extract features from the client vehicle. Instead, the edges on the robot arm were picked up, rendering the tracker failed and unable to resume tracking.



(a) Camera input with no noise **(b)** Camera input with $\sigma = 2$ pixel noise

Figure 2.1: Comparison between original camera image and image with simulated camera noise.

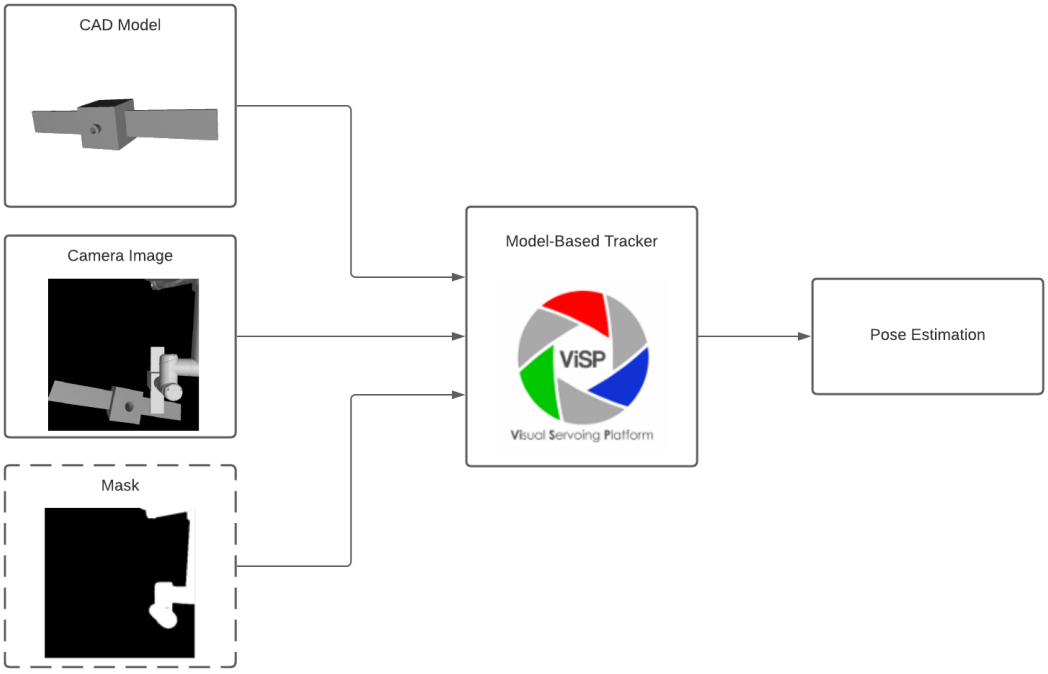
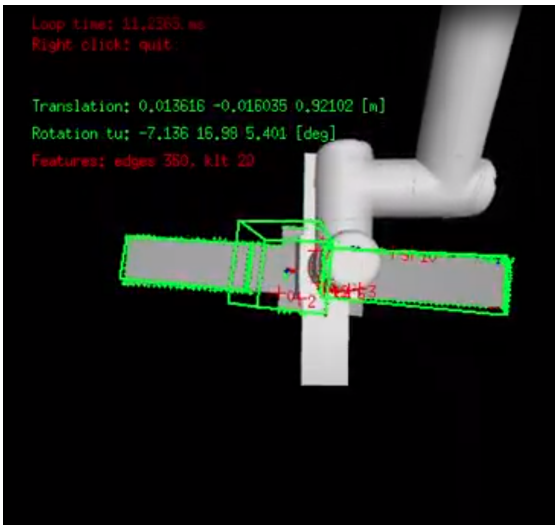
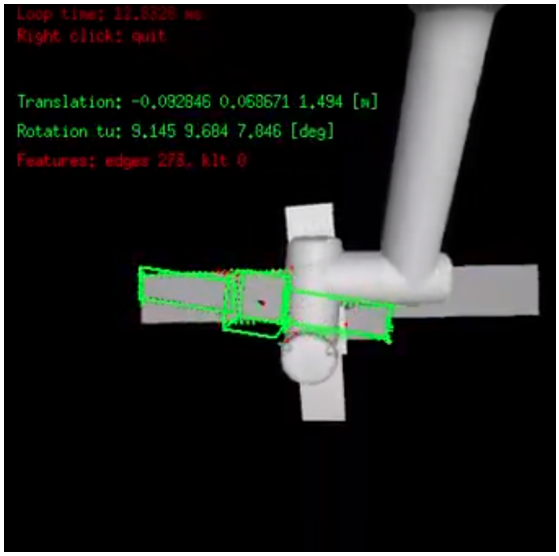


Figure 2.2: ViSP model-based tracker



(a) First fail case of default ViSP tracker



(b) Second fail case of default ViSP tracker

2.2.1 Message passing between ViSP and MuJoCoPy

Because ViSP is only available in C++ library, and we chose MuJoCo under python environment, we need a message passing system with high bandwidth and low latency for fast rate of moderate resolution image. We opt for Lightweight Communication and Marshalling (LCM) [1] to handle the communication between C++ executable and python script. At peak rate, the LCM is transmitting two 400 by 400 grayscale image at around 100Hz with no data loss.

Chapter 3

Hybrid Extended Kalman Filter

3.1 Overview

While ViSP tracker is resilient to image noise and can work under various view angle without occlusion, it can deviate from ground truth pose in just a few frames. Because ViSP does not use any estimation of the object dynamics, it would be helpful to use an Extended Kalman Filter to integrate client vehicle dynamics and provide a better estimate of 6 DoF pose [13].

Because we consider the 6 DoF pose of the client vehicle in the world frame, when there is no contact force on the client vehicle, the dynamical system of the satellite is linear. However, during insertion phase, the MEP can have contact with the client vehicle. This changes the dynamical system to non-linear. Therefore, we implemented a Hybrid Extended Kalman Filter that switches its dynamic model whether a contact force is present.

Predicted state estimate (no contact force)	$\hat{x}_k^- = F\hat{x}_{k-1}^+ + Bu_{k-1}$
Predicted error covariance	$P_k^- = FP_{k-1}^+F^T + Q$
Measurement Residual (no contact force)	$\tilde{y}_k = z_k - H\hat{x}_k^-$
Kalman gain	$K_k = P_k^-H^T(R + HP_k^-H^T)^{-1}$
Updated state estimate	$\hat{x}_k^+ = \hat{x}_k^- + K_k\tilde{y}$
Updated error covariance	$P_k^+ = (I - K_kH)P_k^-$

Table 3.1: Kalman Filter with no contact force (linear dynamics)

Predicted state estimate (contact force)	$\hat{x}_k^- = f(\hat{x}_{k-1}^+, u_{k-1})$
Predicted error covariance	$P_k^- = F_{k-1}P_{k-1}^+F_{k-1}^T + Q$
Measurement Residual (contact force)	$\tilde{y}_k = z_k - h(\hat{x}_k^-)$
Kalman gain	$K_k = P_k^-H_k^T(R + H_kP_k^-H_k^T)^{-1}$
Updated state estimate	$\hat{x}_k^+ = \hat{x}_k^- + K_k\tilde{y}$
Updated error covariance	$P_k^+ = (I - K_kH_k)P_k^-$

Table 3.2: Extended Kalman Filter with contact force (non-linear dynamics)

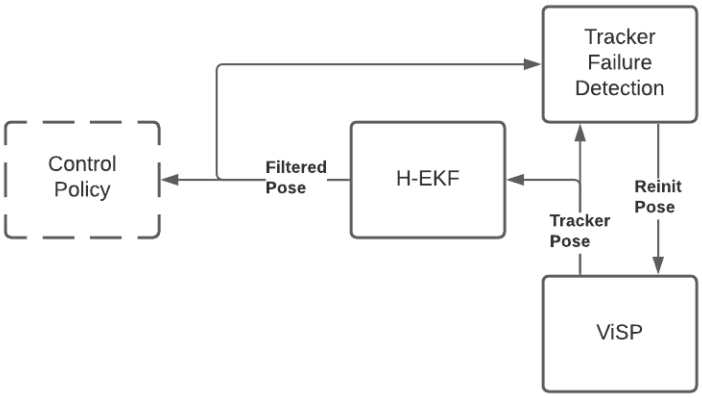


Figure 3.1: Integration of H-EKF with ViSP tracker.

3.2 Integration with ViSP

As shown in Figure 3.1, the H-EKF gets filtered vehicle pose, which is passed to both control policy module and a tracker failure detection module that compares filtered output with tracker output. The detection keeps track of moving average of a sliding window of size of difference between tracker and filtered output. If the average goes beyond a threshold, reset the tracker with filtered pose.

The size of the sliding window s and the threshold value T are chosen empirically. Ill chosen values could lead to either frequent reset of the tracker or deviated reset pose, as the filter have been corrupted by erroneous tracker output.

Chapter 4

Occlusion Handling from joint pose

In Chapter 3 we introduced our use of Hybrid EKF to reinitialize tracker when ViSP lose track of the object. However, when losing track is caused by picking up wrong feature points from occlusion, reinitialized tracker will very soon lose track in one or two frames. This problem cannot be solved without changing the internal function of the tracker. In Chapter 5 we present a methods that handles occlusion by removing the occluded part of the image for feature detection, thus avoiding tracking wrong feature points.

Because we are in a space environment, the only occlusion that could occur in the image comes from robot arm blocking the client satellite. Because we have access to arm geometry and joint position at a given moment, we have a very good estimate of where each link of the robot arm is. Given this information, we can draw the occluded area in the image even without knowing the features in the image.

The benefit of using this method is that it computationally very cheap. Because this method does not rely on image features to estimate occlusion, it computes the occlusion mask without costly feature detection methods like contour detection. As a result, this method can run as fast as the original ViSP tracker. More analysis on computational complexity analysis will be featured in "Results" Chapter.

4.1 Robot Arm projection to image plane

The first step is calculating the 3D position of each joint p_j wrt the base of the UR5e arm base (frame B). Each joint k has a transform matrix $T_k = \left[\begin{array}{c|c} R_k & p_k \\ \hline 0\ 0\ 0 & 1 \end{array}\right]$. This can be computed with the Denavit–Hartenberg parameters given by Universal Robots [3]. The joint coordinates are then transformed into base camera frame (frame A) with transform matrix T_{BA}. The joint coordinates wrt base camera frame is then projected to image coordinates (u, v) from camera matrix $P = \begin{bmatrix} f_x & 0 & w/2 \\ 0 & f_y & h/2 \\ 0 & 0 & 1 \end{bmatrix}$. The camera matrix is relatively simple because the simulated camera has no distortion and the principal point is at the centre. Overall, for each joint k on the

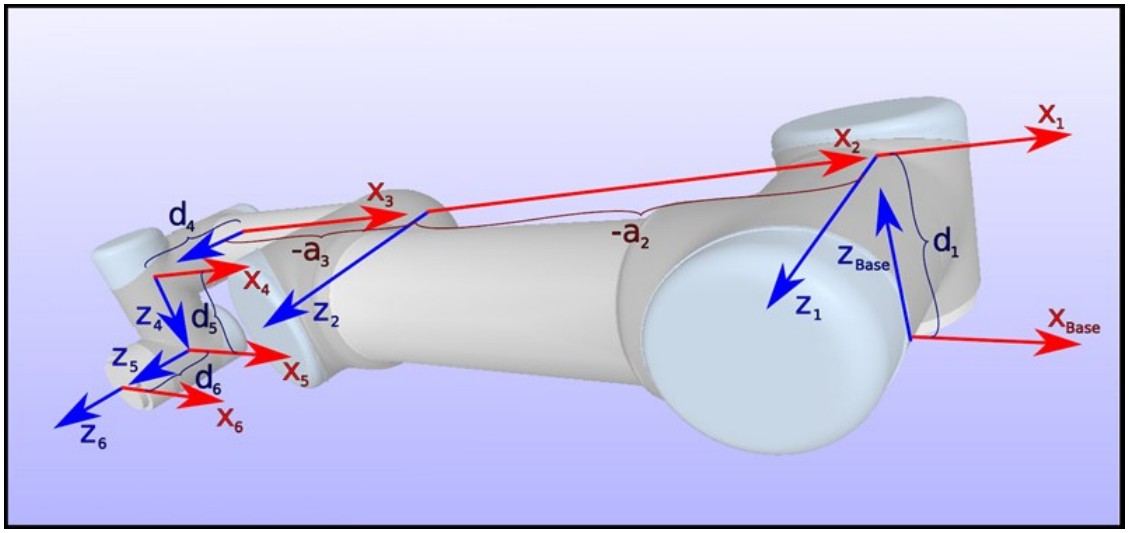


Figure 4.1: UR5e Arm Forward Kinematics

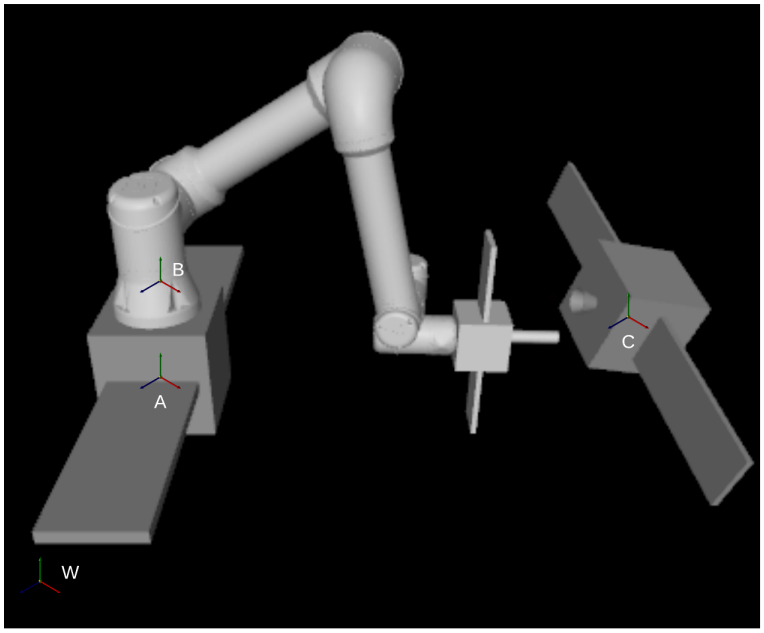


Figure 4.2: Used coordinate frames in simulation environment. W is world frame, A is MRV base frame (camera frame), B is UR5e arm base frame and C is client vehicle frame

arm, the homogeneous 3D coordinate of joint wrt arm base frame V_k is derived from:

$$V_k = \begin{pmatrix} X_k \\ Y_k \\ Z_k \\ 1 \end{pmatrix} = \left[\begin{array}{c|c} R & V_k \\ \hline 0\ 0\ 0 & 1 \end{array}\right] = \prod_{i=1}^{k} T_k \tag{4.1}$$

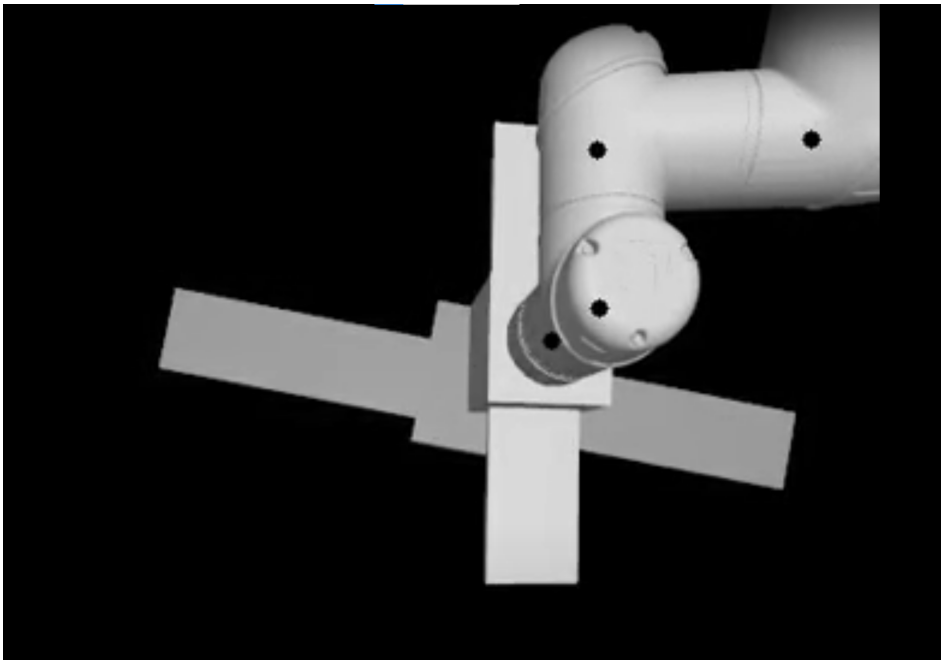


Figure 4.3: Projection of joint coordinate to image

The pixel location of each joint k in the image (u_k, v_k) is derived from:

$$\begin{pmatrix} u_k \\ v_k \\ 1 \end{pmatrix} = P \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} T_{BA} V_k \tag{4.2}$$

The result of the joint projected to image plane can be seen in Figure 4.3. The link can be considered cylinders connecting two joints for the sake of simplification. After the joints are projected to the image pixel positions, the links can be considered line segments with specific width for further simplification. For each specific link, the projection width is $\frac{wf}{z}$, where w is the actual diameter of the UR5e arm, f is the focal length in pixels and z is the distance of link from camera on z axis. In order to completely cover the occluded area, the lines are slightly extended out and the width is set to be slightly wider than the actual pixel width of the link occlusion. Note that we do not mask out the MEP on the end of the arm, because in real world scenario, we may not know the exact dimensions of the MEP.

4.2 Comparison with ground truth occlusion

In Figure 4.4 and Figure 4.5 the original image, the occlusion mask and the occluded results are shown. The mask is capable of blocking most of the arm occlusion in the image and the area of the mask is only slightly larger than the actual occlusion such that object to track is still visible where it's not occluded. However, because the geometry of the arm is not exactly conjoined cylinders, some of the arm are still completely blocked out by the occlusion mask.

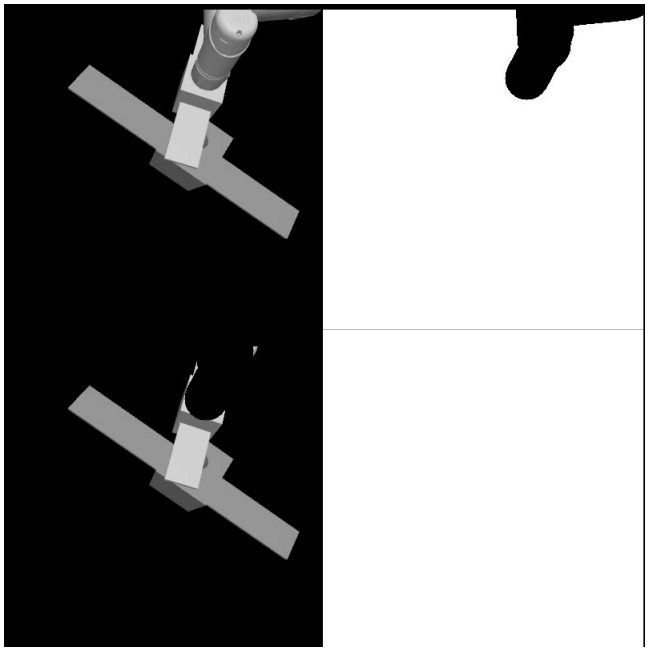


Figure 4.4: Example of occlusion projection, the top left is the original image, the top right is the occlusion mask and the bottom left is the masked image

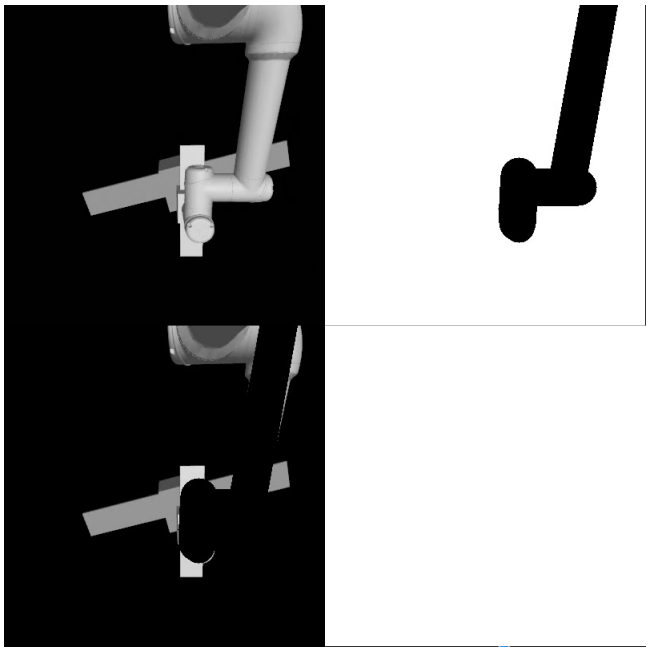


Figure 4.5: Example of occlusion projection

4.3 Integration with ViSP tracker

ViSP tracker needs to be modified to track based on added occlusion information. The mask cannot be directly applied on the image, as the border between the masked region and unmasked

region can be identified by the tracker as feature points. The ViSP tracker, instead of taking in only one image and filtered pose estimate, takes in the original image, a mask of the same size of the image and the filtered pose estimate. Furthermore, because occlusion can block out some features on the client satellite, resulting in less feature points, the ViSP tracker will increase threshold for feature detection when there is a large area of occlusion.

Chapter 5

Occlusion Handling from learning algorithm

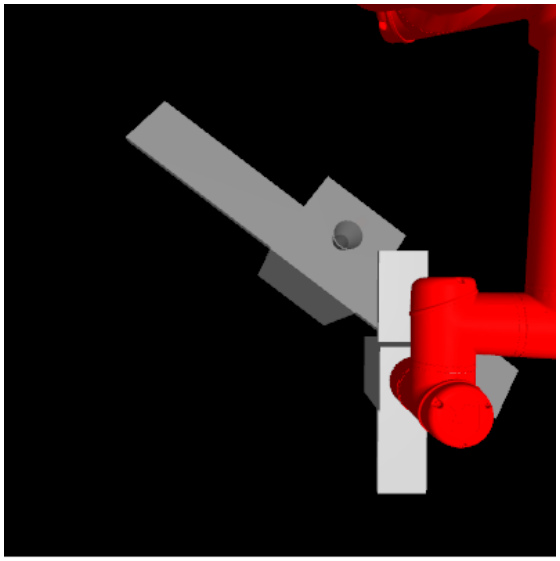
In previous chapter we presented a method of occlusion handling from joint positions. Despite it being relatively robust and computationally cheap, it requires the exact geometry of the arm and its joint position commands. This makes it not readily accessible on a new platform. Also, the occlusion mask generated does not precisely block some of the joint and links, as there are irregular shapes on the arm. In this Chapter, we will look at a learning based method of generating occlusion masks, as the occlusion caused by the robot arm can be identified by an image segmentation network. This method does not require any prior information of the robot arm and generates better occlusion masks.

5.1 Overview

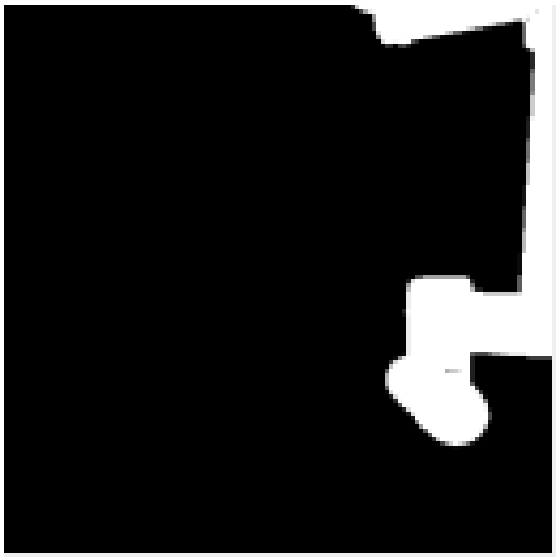
There are various levels of granularity in which the computers can gain an understanding of images. In this scenario, we would like to know if there is occlusion in the frame, and the exact pixel location of the occlusion. Therefore, we would like to classify each pixel as occluded or unoccluded. This task is known as image segmentation. There are two types of segmentation tasks, semantic segmentation and instance segmentation. Semantic segmentation labels each pixel as a corresponding class, while instance segmentation takes a step forward and classifies each instance in the image separately. Because we do not need to discern different instances of occlusion in the frame, we only need semantic segmentation. Common networks used for semantic segmentation includes SegNet [6], UNet[3] and DeepLab [9].

5.2 Data Collection

Because we are using a simulated environment, we can generate large amount of data in an automated fashion without manual operations. To increase variety of training data and include all possibilities, the client vehicle is sampled at random from the possible client positions in Table 2.1, and the initial arm joint angles are sampled from a couple of initial positions. Each



(a) Simulation with different colored arm



(b) Ground truth occlusion mask

sampled initial condition will be ran for two iterations, generating two videos. In both iterations, the arm is controlled by a deterministic controller to complete MEP insertion. In the first iteration, the simulated environment is in its default configurations, and the frames of the output video is the training data. In the second iteration, the color of the arm is changed to red (BGR: (0, 0, 255)), like in Figure 5.1a. The frames in the second video will serve as ground truth mask after simple color extraction. The result looks like Figure 5.1b. Each initial condition will generate around 20 training data/label pairs, and in total around 500 image with ground truth is generated. For simplicity of network architecture and fast training, the images are downsampled to 128 by 128 pixels.

5.3 Network Architecture

The network architecture is a slightly modified version of UNet [19]. The structure of UNet is, as its names suggests, consists of two parts that when combined, shapes like a letter ”U” Figure 5.2. The first section is a conventional convolutional network that extracts image features. In this section, the feature resolution increases and the image resolution decreases. In the second section, it is an upscaling convolutional network that decreases feature resolution and increases image resolution. There is a skip link between each feature extraction layer and upscaling convolution layer of the same level. UNet is suitable for our purpose because of two reasons. First, the skip links consist of saved snapshots of the weights during the first phase of the network and copy them to the second phase of the network. This makes the network combining features from different spatial regions of the image and allows it to localize more precisely regions of interests, resulting in more precise segmentation results. Second, the network is built from the start to output binary classification instead of other multiclass classification network structures, which befits our application.

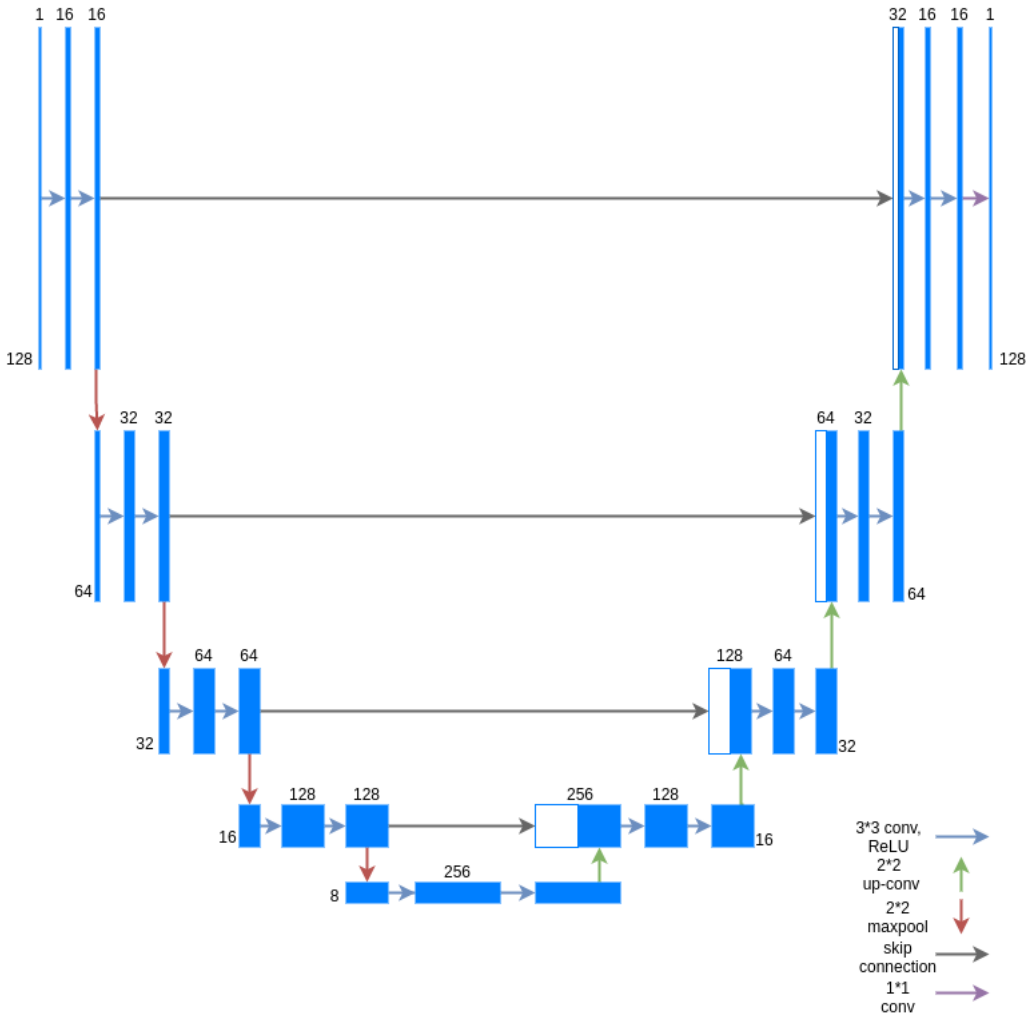


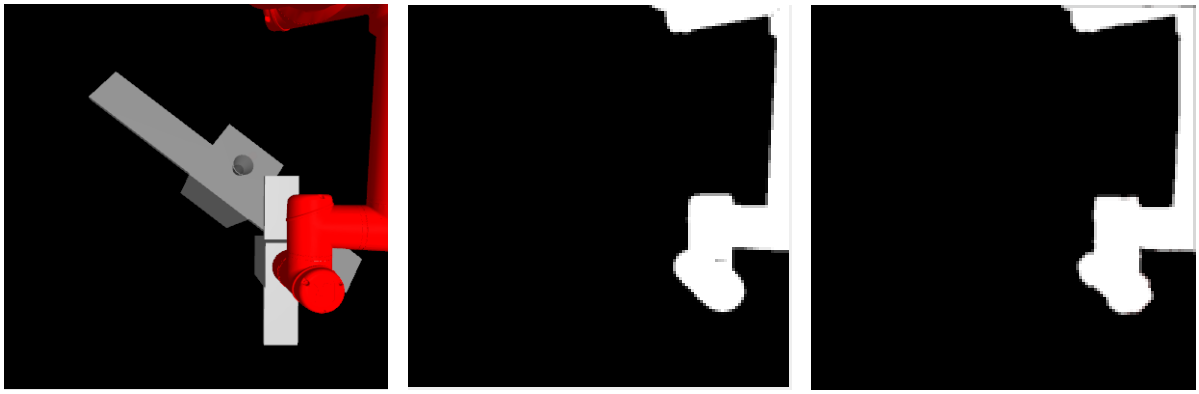
Figure 5.2: UNet Network structure.

Batch size	64
Optimizer	Adam
Optimizer lr	1e-4
Epoch	25

Table 5.1: Parameters used in training

5.4 Training and output

The parameters used during training is shown in Table 5.1. We will not go into detail about hyper parameter tuning and training process as it is not pertinent to the focus of our research. The output from trained network is presented in Figure 5.3 and Figure 5.4. the output occlusion mask from UNet resembles the actual occlusion in the image very well. The general shape is identical and there are no abnormal protrusion inside or outside the masked region. The border of the masked region is not as smooth as the original image, but this is less of a concern, as our



(a) Original image 1 **(b)** Ground truth occlusion 1 **(c)** Trained output occlusion 1

Figure 5.3: Training for occlusion example 1

(a) Original image 2 **(b)** Ground truth occlusion 2 **(c)** Trained output occlusion 2

Figure 5.4: Training for occlusion example 2

tracking algorithm does not rely on these edges for pose estimation.

Chapter 6

Results

In this Chapter, we will be comparing the performance between the original ViSP tracker, the tracker occlusion handling with joint positions and the tracker with occlusion handling with learning method. We will be conducting two sets of experiments. The first set of experiments have the tracker track satellite at different orientation with different levels of occlusion and compare its output with ground truth position. The second set of experiments are integrated tests. The tracker is combined with a controller to test the success rate of the robot arm completing a peg-in-hole insertion task.

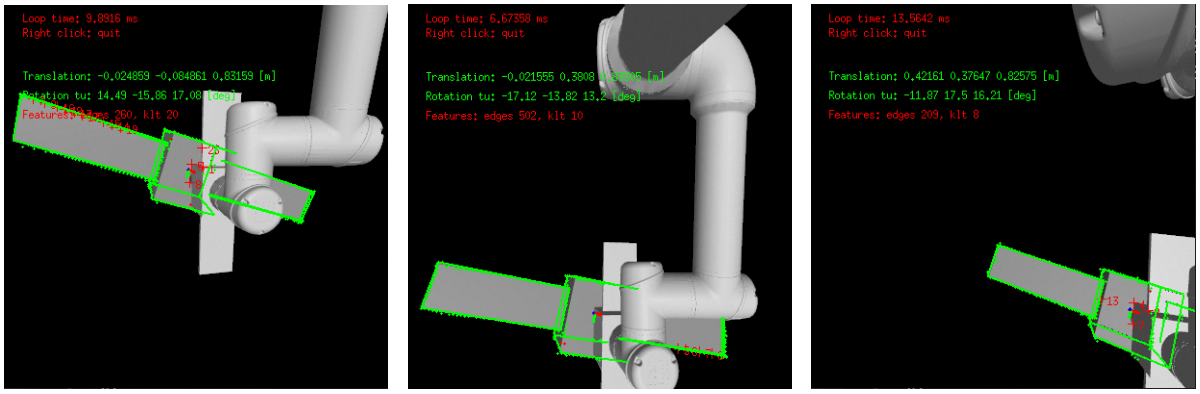
6.1 Accuracy Tests

In this set of tests, we have the the client vehicle with pose sampled from Table 2.1. 3 values are sampled from each field and a total of 729 experiments are conducted. At each experiment, we place the arm directly in front of the client satellite to ensure that there are enough occlusion in the image. Also, the tracker must maintain tracking while the arm is pushed forward to simulate real docking procedure and prove that the tracker is consistent. There are test cases with less occlusion, like in Figure 6.1a and some test cases have larger occlusion like in Figure 6.1b. Some extreme test cases has large part of the satellite out of the frame like in Figure 6.1c.

Table 6.1 is a table of all the tracker performances. In the table, the average and standard deviation error values are calculated after failed cases are removed. The translation error is expressed as Euclidean distance between ground truth and tracker pose. The rotation error is expressed as difference in angle in axis-angle representation.

	Vanilla ViSP Tracker	Occl. from Joint	Occl. from Learning
Success Rate	72%	93%	93%
Avg Trans Err (m)	0.041	0.021	0.021
Avg Rot Err (rad)	0.149	0.082	0.079
Std Trans Err (m)	0.043	0.023	0.022
Std Trans Err (rad)	0.111	0.057	0.052

Table 6.1: Accuracy test of trackers

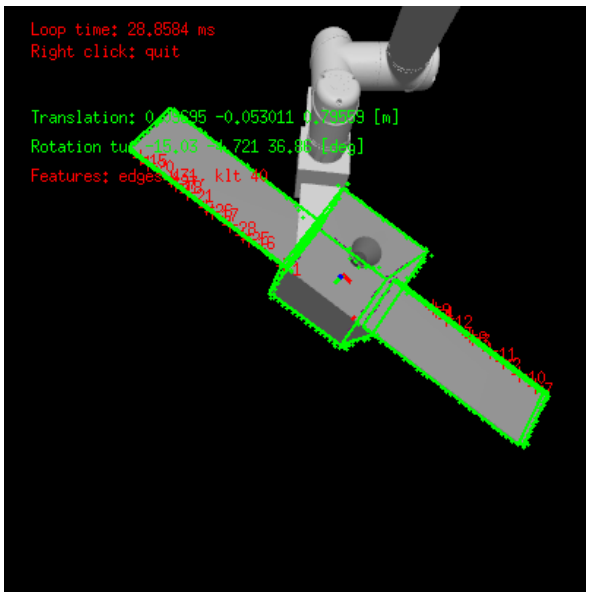


(a) Accuracy test 1 **(b)** Accuracy test 2 **(c)** Accuracy test 3

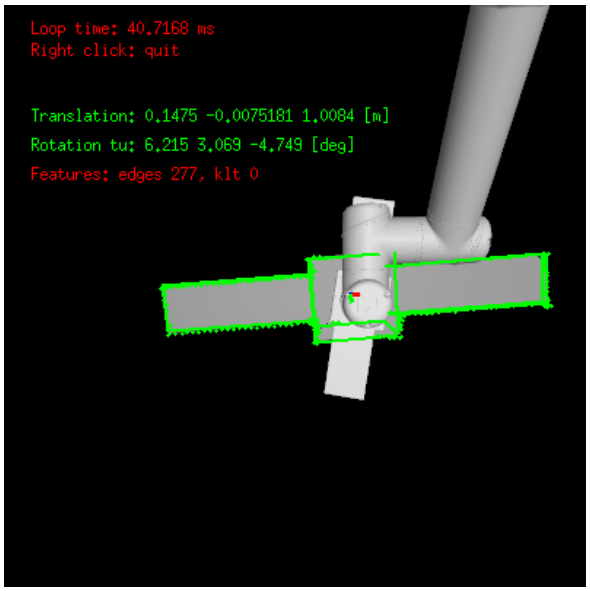
From the results shown in this table, we see that occlusion handling significantly improves success rate and reduces translation and rotation error. It is worth noting that occlusion handling from learning algorithm only improved marginally compared to occlusion handling from joint projection. This is largely due to the fact that the occlusion described by joint position is good enough to remove most erroneous feature points, and when the visible area of the client vehicle is small, like in Figure 6.1c, the tracker performance is limited to the small amount of feature points available to detect.

6.2 Success Rate Tests

Because there are camera noise and ViSP tracker is not deterministic, the tracker can have slightly different outputs given the exact tracked object pose. In this experiment we want to know how robust is the integrated tracker-controller combination. In these tests, the MRV, the robot arm and the client vehicle are positioned at predetermined poses. The tracker output is connected to a deterministic controller that is capable of placing the end effector at the target position. If the tracker is robust and accurate, the controller will be able to complete MEP insertion. If the tracker fails or the tracker is not accurate enough, the arm will miss the target and fail the operation. There are three initial conditions, for each initial condition, we ran 5 batch of tests, each batch 100 times. We collected the success rate stat for each batch and the result is shown in Figure 6.2, Figure 6.3, Figure 6.4. Note that the client vehicle initial pose wrt the MRV is a 6 element array consisting of [x, y, z, roll, pitch, yaw] and the robot arm initial joint angles is a 6 element array consisting of joint angles starting from the base joint to the last wrist joint. The translations are measured in meter and the rotations are measured in radian. The success rate of our occlusion handling tracker, whether from joint position or learning, is consistently above 95%. In comparison, the original ViSP tracker has performance between 70% and 80%.



(a) Image of initial condition 1

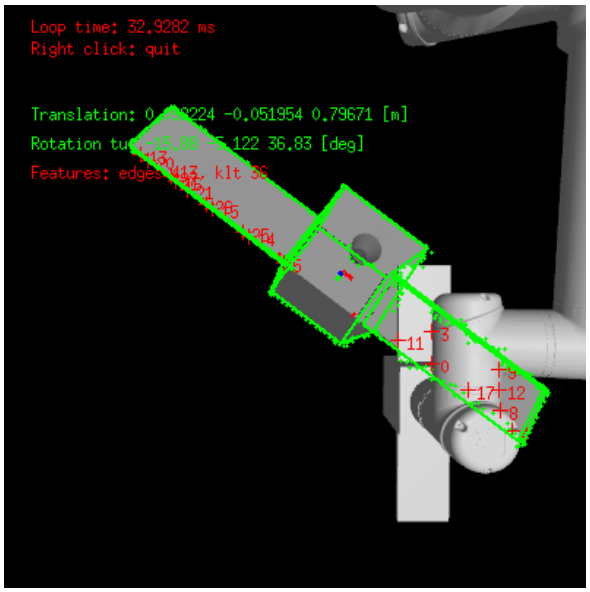


(b) Image of final condition 1

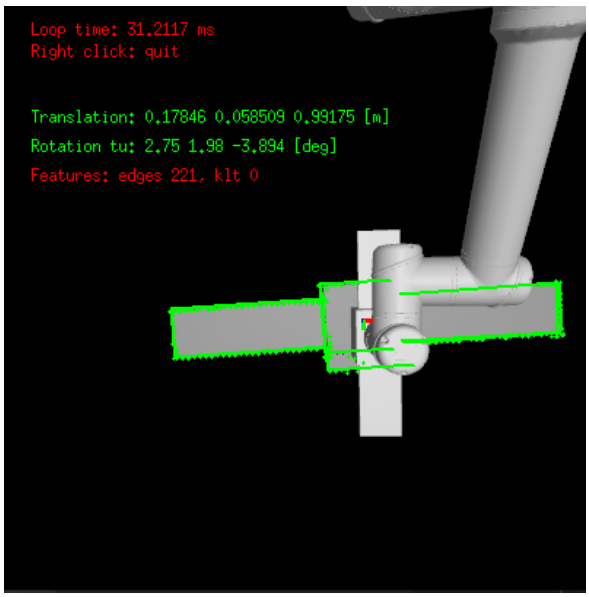
	Original Tracker	**Occl. from Joint**	**Occl from Learning**
Success Rate	$79.8 \pm 2.7\%$	$97.6 \pm 1.0\%$	$97.8 \pm 0.7\%$

(c) Client Vehicle Initial Pose: [0.8, 0.1, 0.2, 0.65, 0.3, 0.0]
Robot Arm Joint Initial Angle: [0.0, -0.8, 0.8, 0.3, 1.5, -1.5]

Figure 6.2: Test with initial condition 1



(a) Image of initial condition 2

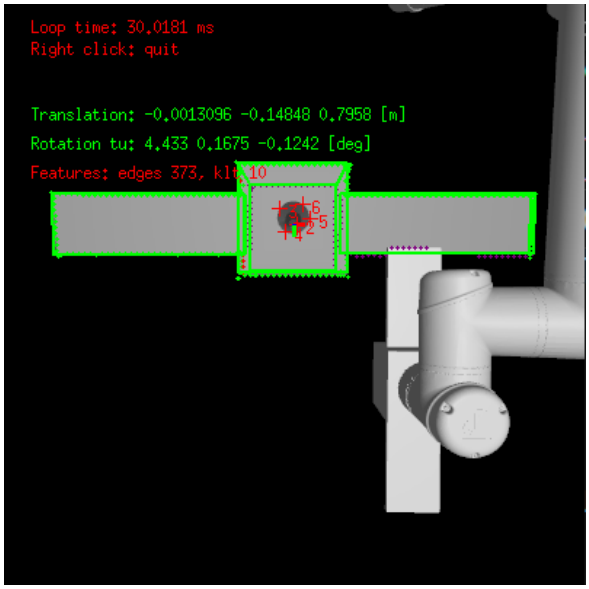


(b) Image of final condition 2

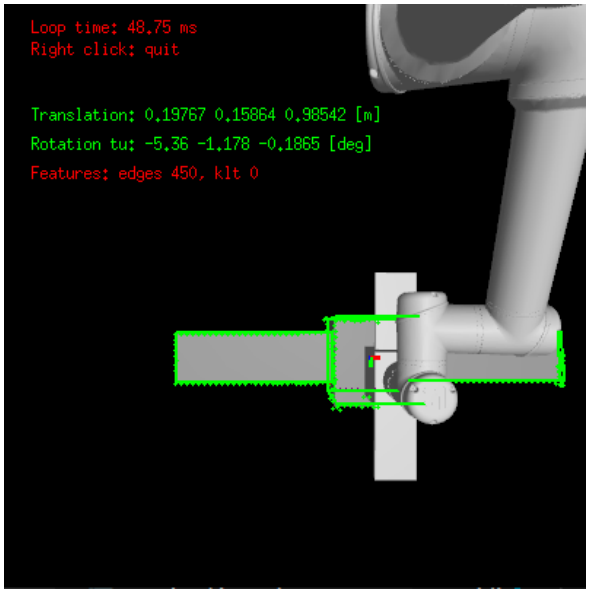
	Original Tracker	**Occl. from Joint**	**Occl from Learning**
Success Rate	$79.4 \pm 4.3\%$	$97.4 \pm 0.8\%$	$97.8 \pm 1.3\%$

(c) Client Vehicle Initial Pose: [0.8, 0.1, 0.2, 0.65, 0.3, 0.0]
Robot Arm Initial Joint Angle: [-0.3, -0.5, 1.9, -1.4, 1.3, -1.6]

Figure 6.3: Test with initial condition 2



(a) Image of initial condition 3



(b) Image of final condition 3

	Original Tracker	**Occl. from Joint**	**Occl from Learning**
Success Rate	$73.2 \pm 4.0\%$	$96.4 \pm 1.4\%$	$96.4 \pm 0.8\%$

(c) Client Vehicle Initial Pose: [0.8, 0.2, 0.3, 0, -0.1, 0.0]
Robot Arm Initial Joint Angle: [-0.3, -0.5, 1.9, -1.4, 1.3, -1.6]

Figure 6.4: Test with initial condition 3

	Mask generation	ViSP execution	Overall runtime
Original ViSP Tracker	0ms	10ms-15ms	10ms-15ms
Occl. from Joint	0.5ms	11ms-15ms	11.5ms-15.5ms
Occl. from Seg. (GPU)	40ms	11ms-15ms	51ms-55ms
Occl. from Seg. (CPU)	480ms	11ms-15ms	500ms

Table 6.2: Process time of each frame of each methods

6.3 Runtime Analysis

Because we do not have the exact computational hardware setup as NGC, we cannot provide definitive results of our pose estimation's runtime on GC hardware. However, given the performance of NGC hardware on original ViSP tracker, we can have a good estimate of our methods' performance on NGC computational hardware.

The original ViSP tracker runs at around 5Hz on NGC hardware. This means the process time of each frame is around 200ms. By comparison, the process time of each frame of original ViSP tracker on our computer is around 10ms to 15ms depending on the image input. In Table 6.2 are the run time results of all the trackers. Note that the results from image segmentation network deployed on CPU are not representative of the real performance, as the CPU is also running simulation, which is a computationally heavy task and consumes available memory for inference, and the model is not optimized for CPU inference [17].

The occlusion handling from joint method is only around 8% slower than the original ViSP tracker. If we assume that the run time for each frame scales linearly from our hardware to NGC hardware, the process time of each frame is 216ms. That translates to around 4.5Hz of pose estimation output, which is well within the requirements at NGC side (2-5Hz).

Unfortunately, the current state of occlusion handling from segmentation deployed on GPU cannot function on NGC hardware and occlusion handling from segmentation deployed on CPU requires much CPU RAM space and would not work well on NGC hardware. However, these methods can be useful when there is GPU support in the future, as these methods are still faster than other learning based works. In comparison, an implementation of [11] is around 40% slower on our hardware.

Chapter 7

Related Works

7.1 Computational Complexity

With advancement of GPU hardware acceleration, works in the last 5 years have been largely focusing on using more elaborate network structure for feature extraction or even full tracking in an end-to-end fashion. Most of the works focus on tracker with high accuracy or tracker that deal with multiple object tracking, but few works focus on pose estimation with low computational complexity. [22] is one of the few works that focus on fast pose estimation without the use of GPU. The major contribution of this work is an edge-based approach which provides comparable precision to the state of the art with less hardware requirements. One part of particular interest to us is the occlusion detection. In this work, the occlusion is detected by looking at the contour of the object and the tracking result is also robust against occlusion. Compared to our approach, this method does not rely on any geometry of occlusion. However, the occlusion is not reliably detected. Because the contour detection is not entirely accurate, it can sometimes draw a contour that is not representative of the actual contour of the object, thus reducing the accuracy of its occlusion detection.

Another work [8] proposes two lightweight neural network structure that can even run on CPU on smart phone platform. This is a significant influence of our work's neural network architecture, as our network is also a simplified, lightweight version of the original network. However, due to time constraints we are unable to implement CPU inference optimization in our work. With CPU inference optimization [17], it is possible to speed up the inference time on CPU 10 times and enable running our model on CPU real time.

7.2 Selecting Region of Interest

Many related works on 6 DoF Benchmark for 6D Object Pose Estimation (BOP) challenge [14]. The goal of BOP is to capture the state of the art in estimating the 6D pose, i.e. 3D translation and 3D rotation, of rigid objects from RGB/RGB-D images. Similar challenges exist in our setup and the BoP. For instance, the majority of data comes from simulated environment and the target object can be partially occluded. In the BoP 2020 challenge, there are two works that base on the idea of generating a region of interest (RoI) first and tracking within the RoI for

faster tracking and higher accuracy. In [15] the authors uses a hybrid of feature matching and deep learning instance segmentation. The instance segmentation picks out object to track within a target area and conventional image feature matching method estimates the pose of the object. The performed segmentation improves tracking accuracy by removing occlusions and noise such as shadows, but also reduce the area of feature matching, therefore increasing runtime speed. [16] has similar ideas of using a 2D object detection network to pick out the RoI and estimate object pose in a smaller region. Our work also used similar idea of applying a mask before pose estimation. [15] and [16] draw a RoI to aid pose estimation and our work draw a ”Region of Exclusion”, where the masked region is avoided. The major difference is their RoI comes from neural network output that is trained on all the objects that you need to track, whereas in our work, the mask does not rely on the object to be tracked and if the arm geometry is the same it requires no tuning or training for tracking different objects.

Chapter 8

Conclusion and Future works

This work builds a 6 DoF visual pose estimation framework that interacts with simulation and presents two methods of handling occlusion in visual pose estimation in a space environment. Both methods use occlusion mask to prevent visual tracker from extracting erroneous feature points, and both methods use a hybrid Extended Kalman Filter to improve tracking results, re-initialize tracker when necessary and report failure. The first method uses joint position from forward kinematics to estimate an occlusion mask. The second method uses deep learning image segmentation to output an occlusion mask given only input image. The first method creates reliable occlusion masks and can operate at high frequency with low computational power. The second method creates a finer occlusion mask and does not rely on arm geometry. Finally, we prove that both methods significantly improve tracking accuracy and reduce rate of losing track over the original visual tracker.

There are two major points that can be worked on in the future. The first one is tracking moving object with occlusion. The object to be tracked in our set up is relatively stationary wrt MRV base. If we can modify the method to track object that is moving at relatively higher speed, we can use it in other environments. The second one is transferring current result to real hardware. Currently all evaluations happen within software simulation, and there are many influences yet to be considered when transferring the method to real world setup, like dynamic lighting condition and environment with realistic texture.

Bibliography

[1] Lightweight communications and marshalling. https://lcm-proj.github.io/. 2.2.1

[2] Mujoco website. http://www.mujoco.org/. 2.1

[3] Dh parameters for calculations of kinematics and dynamics. https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/. 4.1, 5.1

[4] Visp website. https://visp.inria.fr/, . 2.2

[5] Visp website. https://visp.inria.fr/mbt/, . 2.2

[6] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation, 2016. 5.1

[7] Quentin Bateux, Eric Marchand, Jürgen Leitner, Francois Chaumette, and Peter Corke. Visual servoing from deep neural networks, 2017. 1.2

[8] Bowen Chen, Juhan Bae, and D. Mukherjee. Fast 6dof pose estimation with synthetic textureless cad model for mobile applications. *2019 IEEE International Conference on Image Processing (ICIP)*, pages 2541–2545, 2019. 7.1

[9] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, 2017. 5.1

[10] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4397–4404, 2015. doi: 10.1109/ICRA.2015.7139807. 2.1

[11] Mathieu Garon and Jean-Francois Lalonde. Deep 6-dof tracking. *IEEE Transactions on Visualization and Computer Graphics*, 23(11):2410–2418, Nov 2017. ISSN 1077-2626. doi: 10.1109/tvcg.2017.2734599. URL http://dx.doi.org/10.1109/TVCG.2017.2734599. 1.1, 2.1.2, 6.3

[12] Brent A. Griffin, Victoria Florence, and Jason J. Corso. Video object segmentation-based visual servo control and object depth estimation on a mobile robot, 2020. 1.2

[13] Kjartan Halvorsen, Torsten Söderström, Virgil Stokes, and Håkan Lanshammar. Using an Extended Kalman Filter for Rigid Body Pose Estimation. *Journal of Biomechanical Engineering*, 127(3):475–483, 12 2004. ISSN 0148-0731. doi: 10.1115/1.1894371. URL https://doi.org/10.1115/1.1894371. 3.1

[14] Tomas Hodan, Martin Sundermeyer, Bertram Drost, Yann Labbe, Eric Brachmann, Frank Michel, Carsten Rother, and Jiri Matas. Bop challenge 2020 on 6d object localization, 2020. 7.2

[15] Rebecca König and Bertram Drost. A hybrid approach for 6dof pose estimation. In Adrien Bartoli and Andrea Fusiello, editors, *Computer Vision – ECCV 2020 Workshops*, pages 700–706, Cham, 2020. Springer International Publishing. ISBN 978-3-030-66096-3. 7.2

[16] Jinhui Liu, Zhikang Zou, Xiaoqing Ye, Xiao Tan, Errui Ding, Feng Xu, and Xin Yu. Leaping from 2d detection to efficient 6dof object pose estimation. In Adrien Bartoli and Andrea Fusiello, editors, *Computer Vision – ECCV 2020 Workshops*, pages 707–714, Cham, 2020. Springer International Publishing. ISBN 978-3-030-66096-3. 7.2

[17] Yizhi Liu, Yao Wang, Ruofei Yu, Mu Li, Vin Sharma, and Yida Wang. Optimizing CNN model inference on cpus. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 1025–1040, Renton, WA, July 2019. USENIX Association. ISBN 978-1-939133-03-8. URL `https://www.usenix.org/conference/atc19/presentation/liu-yizhi`. 6.3, 7.1

[18] E. Marchand, F. Spindler, and F. Chaumette. Visp for visual servoing: a generic software platform with a wide class of robot control skills. *IEEE Robotics Automation Magazine*, 12 (4):40–52, 2005. doi: 10.1109/MRA.2005.1577023. 2.2

[19] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. 5.3

[20] Riccardo Spica, Paolo Giordano, and François Chaumette. Coupling visual servoing with active structure from motion. 05 2014. 1.2

[21] David Joseph Tan, Federico Tombari, Slobodan Ilic, and Nassir Navab. A versatile learning-based 3d temporal tracker: Scalable, robust, online. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 693–701, 2015. doi: 10.1109/ICCV.2015.86. 1.1

[22] Lucas ValenÇa., Luca Silva., Thiago Chaves., Arlindo Gomes., Lucas Figueiredo., Lucio Cossio., Sebastien Tandel., João Lima., Francisco Simões., and Veronica Teichrieb. Real-time monocular 6dof tracking of textureless objects using photometrically-enhanced edges. In *Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 5: VISAPP,*, pages 763–773. INSTICC, SciTePress, 2021. ISBN 978-989-758-488-6. doi: 10.5220/0010348707630773. 7.1