# A Robot Who Knows How to Take the Elevator

A project of the 2019 Robotics Course of the School of Information Science and Technology (SIST) of ShanghaiTech University
https://robotics.shanghaitech.edu.cn/teaching/robotics2019

**Shenhan Qian, Xianing Chen, Zehao Yu**
{qianshh,chenxn1,yuzh}@shanghaitech.edu.cn

Dec 2019

**Abstract**

Recent research of indoor autonomous robot have shown great success, but most of these work focus on one floor environment. As a result, current solution of indoor robots can not directly apply to across floor environment. To close the gap between one floor environment and complex multi-floor environment, we propose an elevator robot that can find an elevator and take the elevator to different floor on its own and thus enables the robot to work across multi-floor environment. Our robot is built upon a robust four wheeled platform powered by electric motors. It is equipped with color and depth cameras, laser scanners, inertial measurement units to locate objects and itself. We conduct experiments to test our robot, including navigation, button detection, and manipulation in SIST.

# 1 Introduction

With the rapid development of computer hardware and software, autonomous robot is no longer a dream. There are so many autonomous robots showing their capability to perform complex tasks. For instance, Boston Dynamics's ATLAS robot can even turn a somersault. Motivated by these impressive demo, we also want build our own robot. Our project is about build a robot which can take the elevator on its own. Although it seems like a simple task, it's a huge step towards general autonomous robot. The robot is expected to perceive the environment which means it has some understanding of its surroundings and knows where it is. It also needs to find its way to the elevator, then push the button with its arm, wait until the elevator's door opens, and finally drive in. This process requires ability of mapping, localization, navigation, visual object detection and manipulation. With such general ability, we believe that we can build an even powerful robot based on it.

# 2 State of the Art

## 2.1 Shenhan Qian - Localization and Mapping

Simultaneous localization and mapping (SLAM) is a computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. SLAM algorithms are tailored to the available resources, hence not aimed at perfection, but at operational compliance.

- **EGO-SLAM: A Robust Monocular SLAM for Egocentric Videos** Patra et al. [9] proposed to solve SLAM as an SFM problem over the sliding temporal windows, initializing the camera poses using 2D rotation

averaging, followed by translation averaging before structure estimation using bundle adjustment. This helps in stabilizing the camera poses when 3D estimates are not reliable.

- **Beyond Photometric Loss for Self-Supervised Ego-Motion Estimation**
  To acquire accurate relative pose, previous works rely on the photometric error generated from depths and poses between adjacent frames, which contains large systematic error under realistic scenes due to reflective surfaces and occlusions. Shen et al. [11] tried to bridge the gap between geometric loss and photometric loss by introducing the matching loss constrained by epipolar geometry in a self-supervised framework.

- **Network Uncertainty Informed Semantic Feature Selection for Visual SLAM**
  Ganti and Waslander [4] present SIVO (Semantically Informed Visual Odometry and Mapping), a novel information-theoretic feature selection method for visual SLAM which incorporates semantic segmentation and neural network uncertainty into the feature selection pipeline. Our algorithm selects points which provide the highest reduction in Shannon entropy between the entropy of the current state and the joint entropy of the state, given the addition of the new feature with the classification entropy of the feature from a Bayesian neural network. Each selected feature significantly reduces the uncertainty of the vehicle state and has been detected to be a static object (building, traffic sign, etc.) repeatedly with a high confidence. This selection strategy generates a sparse map which can facilitate long-term localization.

- **ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras**
  ORB-SLAM2 [8] is a complete SLAM system for monocular, stereo and RGB-D cameras, including map reuse, loop closing and relocalization capabilities. The system works in real-time on standard CPUs in a wide variety of environments from small hand-held indoors sequences, to drones flying in industrial environments and cars driving around a city. The method's back-end based on bundle adjustment with monocular and stereo observations allows for accurate trajectory estimation with metric scale. The system includes a lightweight localization mode that leverages visual odometry tracks for unmapped regions and matches to map points that allow for zero-drift localization. The system has three main parallel threads:

  1. The tracking to localize the camera with every frame by finding feature matches to the local map and minimizing the reprojection error applying motion-only BA(Bundle Adjustment).
  2. The local mapping to manage the local map and optimize it, performing local BA.

3. The loop closing to detect large loops and correct the accumulated drift by performing a pose-graph optimization. This thread launches a fourth thread to perform full BA after the pose-graph optimization, to compute the optimal structure and motion solution.

## 2.2 Zehao Yu - Navigation

Navigation is the problem of moving a robot from one place to a target place according to some instruction within some environment. It's the basis of many real world mobile robotic application such as house cleaning, patrolling, package delivery and of course elevator robot. While there are huge literature of this area, we briefly review some recent learning based method on navigation.

- **Learning Navigation behaviors End-to-End With AutoRL**

  Chiang et al. [2] propose to learn end-to-end point-to-point path-following navigation behaviors that avoid moving obstacles using Auto Reinforcement Learning (AutoRL). The policies take as input noisy lidar measurement and output linear and angular velocities directly. AutoRL enables learning such policies that have been difficult or impossible to program manually in a end-to-end manner. The evaluation result in the paper shows the learned policies not only generalize to new environments and moving abstacles but also are robust to sensor, actuator and localization noise.

- **Don't Make the Same Mistakes Again and Again: Learning Local Recovery Policies for Navigation from Human Demonstrations**

  Del Duchetto et al. [3] present a human-in-the-loop learning framework for mobile robots to generate local policies in order to recover from failures. In their framework, a failure classifier is trained and was used to identify failure case. When a failure is detected, a human operator confirms the failure and demonstrates how to recover from that situation.

- **Reinforced Cross-Modal Matching and Self-Supervised Imitation Learning for Vision-Language Navigation**

  Vision-language grounded navigation aims at carry out natural language instruction and have received increased attention. Wang et al. [12] propose a reinforced cross-model matching approach to encourage global matching between instructions and trajectories. The authors also introduce a Self-supervised Imitation Learning method for the agent to explore unseen environments by imitating its own past, good decisions.

- **Sim-Real Joint Reinforcement Transfer for 3D Indoor Navigation**
  Since most of the learning based methods was trained in some synthetic environment, there is a domain gap when we deploy the trained agent in real world. First, the visual representation of synthetic environment and real world environment have significant variances. Second, the house plane is different. Therefore, Zhu et al. [13] propose to jointly adapt visual representation and policy behavior to leverage the mutual impacts of the environment and policy. In particular, the agent policy network is trained on a large scale synthetic data because such data is easier to obtain. Then a feature adaption step is used to map the visual input in real environment to the same latent space with synthetic environment such that the policy network trained on the synthetic environment can be directly applied. Furthermore, an auxiliary approach (policy distillation) is also proposed to close the gap between two domain. The model learned from synthetic environment acts like a teacher and a student model learns the knowledge from it.

## 2.3 Xianing Chen - Detection and Overall design

Object detection is an important part of robot interaction with outside world. Moreover, with the computation limitation of robot, speed and lightweight model is at the heart of the design. Also, robot must work with hardware to meet the challenges of the external environment, especially in our indoor environment. So, the overall design is important for our robot.

- **ThunderNet: Towards Real-time Generic Object Detection**
  Previous CNN-based object detectors suffer from enormous computational cost, which hinders them from real-time inference in computation-constrained scenarios especially in the field of robotics. So Qin et al. [10] present a fast and efficient lightweight detector network called ThunderNet with a lightweight backbone designed for object detection and an extremely efficient RPN and detection head design. Also they design two efficient blocks, Context Enhancement Module and Spatial Attention Module to generate more discriminative feature representation.

- **Detection-Tracking for Efficient Person Analysis: The DetTA Pipeline Stefan**
  In this paper,Breuers et al. [1] combine detection, tracking and CNN-based analyzer into a real-time and lightweight fully modular detection-tracking-analysis pipeline which are used by robots. They research the application of person analysis, and this design can be easily expanded to button condition. By combine those components, the robot can analyze button status accurately.

- **PVANET: Deep but Lightweight Neural Networks for Real-time Object Detection**
  Kim et al. [6] presents a way to achieve the state-of-art-accuracy in detection task with very deep network while minimizing the computational cost by redesigning the feature extraction part and adopting some building blocks including concatenated ReLU, Inception and HyperNet.

- **Design of an Autonomous Robot for Mapping, Navigation, and Manipulation in Underground Mines**
  Traditional robots, sensors and software often do not work reliably underground due to the harsh environment, so Lösch et al. [7] analyses requirements and presents a robot design capable of navigating autonomously underground and manipulating objects with a robotic arm.
  The robot's base is a robust four wheeled platform powered by electric motors and able to withstand the harsh environment. It is equipped with color and depth cameras, lights, laser scanners to overcome the dark underground environment, so it can finetunes created map and navigates autonomous in underground mines. For its manipulation, the robot is equipped with a robotic arm UR5 and a 3-Finger Adaptive Robot Gripper from Robotiq. And in order to improve water and dust resistance, they were retrofitted with a rubber "sleeve", which only minimally decreases range of motion. Besides, it has an inertial measurement unit, two dedicated computers, a potent battery and so on to assist in its main task. As a result, the robot can be operated by remote control, autonomously drive through the mine by following waypoints on a map, or autonomously explore unknown terrain.

## 2.4 One More Paper

**MoveIt! Task Constructor for Task-Level Motion Planning**
Motion planing aims at find trajectories between a start point to a goal point. It is a fundamental problem in robotics and of great importance. When the planing consists of multiple interdependent sub-tasks, it becomes extreme difficult. However, little work have been done for this purpose. Therefore, Görner et al. [5] presents a task constructor framework to provides a flexible and transparent way to plan a complex task. In their framework, each sub-task is solved in isolation with a black-box planning stages and then common interface is used to combine the different solution hypotheses. Furthermore, this framework is very flexible because the sub-tasks can be organized to form a hierarchical structure which be solved in parallel or sequentially. In our project, the robot needs to solve two main task: plan the path to elevator and use the arm to push the button. Our task is straightforward. We first plan the path to elevator then the robot will stop in front of the button. Then we use *MoveIt* to plan the motion of the arm in order to push the button.

# 3 System Description

The hardware system is built upon a Jackal robot, which is a move base. A 3D laser scanner is mounted on Jackal to enable mapping and localization. A robot arm with an RGB-D camera mounted on its end is installed so that elevator button detection and pushing are possible.

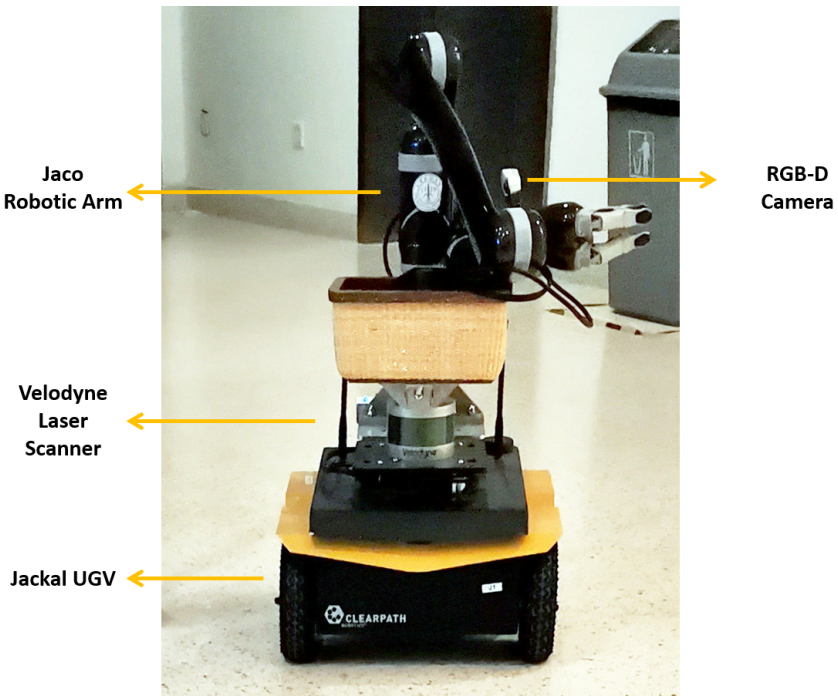


Figure 1: Hardware components of our robot.

All the devices mentioned above are compatible with ROS(The Robot Operating System), and our software framework is built upon ROS as well. We will use some of the powerful software packages of ROS, such as *Gmapping* for SLAM, *Moveit* for button pushing, and *Smach* and *Navigation Stack* for task planing.

With the integration of the software and hardware system, our robot is expected to complete the following tasks without human interference. With a predefined map in its memory, the robot should plan its path and navigate towards an elevator under the help of a laser scanner. Upon approaching the elevator, our robot needs to detect where the button is and use its arm to push the button. In this process, the RGB-D camera on the end of the arm will help determine the precise 3D coordinate of the button. When the elevator's door opens, the robot should directly drive in.

### 3.1 Technical Details and Main Problems

- **Wireless Communication**
  To exchange data between the robot and the laptop, it is necessary to setup a wireless communication. The first choice is to respectively connect the robot and laptop to the public hot spot, and transfer data upon LAN(Local Area Network). However, since the bandwidth is limited, network delay is intense especially when running graphic programs. Then we tried to setup up a hot spot upon the wireless module of a laptop and let the computer on the robot directly connect to the hot spot of the laptop, in additional with a compressed way of ssh connection, it is possible to run an Rviz client while the robot is moving.

  Sooner or later, we were exposed to a better way to run ROS on multiple machines. We set the jackal robot to be the master and our laptops be slaves, transfer data online while rendering windows locally. This method requires much small band width and works smoothly and robustly.

- **Navigation**
  In order to let the robot go to the elevator, we use Jackal's built-in particle filter based algorithm to implement localization and navigation. The first step is to use gmapping to create the indoor map. After we load the created map we send goal position topics to navigate package to make our robot automatic go to the goal point. However, the navigation precision is low because of the limitation of the laser scanner and the estimated initial pose error. So Jackal will still arrive the setting goal with a relatively intolerable error. This will seriously affect the transition between each task state. On the other hand, the route planing algorithm is not robust enough, this makes robot sometimes entrap a collision point. To solve this problems, we decompose the path to a series of goal points which is easy for the navigation algorithm. Then the robot can successfully achieve the goal position but still with a relative low localization precision.

- **Button Detection**
  The goal of this part is to find the 3D position of the button with respect to the camera. Our pipeline is to first detect the button in an RGB image, then align its 2D coordinate with the depth map so that the 3D coordinate is available.

  In order to detect the buttons of an elevator in an RGB image, we first collected pictures of them and analyzed their features. We found it hard to define the buttons simply using edges or colors. Therefore, we adopted the template-matching algorithm for button detection. The algorithm uses a predefined image patch called template to perform convolution operation on the input image to obtain an activation map. Where the activation value is high, where our objective might occur with high possibility. We cropped out background of the collected pictures and generated templates for each button.

Given the 2D coordinate and its corresponding depth value, it is easy to calculate the 3D coordinate, which is an inverse process of pinhole camera imaging.

The template-matching algorithm works generally well, but it has several limitations. First, since a template is essentially a cropped image, vanilla template-matching doesn't have the property of scale-invariance. One has to tune the size of this template to achieve an optimal detection performance. In order to achieve scale-invariance, image pyramid is an option. Second, this algorithm suffers from degradation when lighting condition changes, mistakes appear more often at night. What's more, since the button and the console of the buttons are made of either transparent plastic or metal, reflection of intensive light source influences our detection result.

- **Manipulation**
  In this part, we describe how to use the robot arm to push the button. Here we assume the button has been detected and its 3D position in the camera frame has been published. In order to transform the 3D position in the camera frame to root frame of the arm, we need to do hands-eye calibration. However, as our camera is fixed onto the arm, only when the camera if facing the button at about same height, the button can be captured in the FOV of the camera. So we need to fix pose of the arm such that the camera can detect the button at a high precision. With the above consideration, we fix the arm pose at the beginning of the button detection. To implement this part, we set an initial pose of the arm.

  When the arm is ready, we keep listening to the button detection result. After receiving the button detection result at a stable stage, we can control the arm to push the button. However, we can't directly control the finger of the arm, what can be controlled is the end-effector of the arm which is the last joint of the arm. Therefore, we need to transform the movement of the finger to the movement of the end-effector. We assume that the relative position of the finger to the end-effector of the arm is fixed. As a consequence, the transformation of the finger and the transformation of the end-effector is the same. So we only need to find out the transformation from the finger to button. As we mount the camera at the end-effector of the arm, the finger is visible in the camera and the transformation can be easily obtained. So we pre-compute the 3D position of the finger in the camera frame and when the 3D position of the button arrives, we get the transformation of the end-effector by a simple subtraction.

  After we getting the transformation of the arm, we can use *Moveit* to control the *Kinova-arm*, the implementation is straightforward. In particular, we first plan the trajectory of the arm and execute it. When the arm has pushed the button, we reset it to the initial pose. However, constraints of the trajectory are required, otherwise, at some cases **the arm can not move smoothly to the button**. Unfortunately, with such constraints, the planning will fail within a predefined planning time constraints (5

seconds).

- **State Machine**
  In order to make the robot complete tasks automatically, we use Smach to control the transition between task states.

  The pipeline of our state machine is first loading the map's key points positions. Then check each module's working state to ensure the robot can work well. After that, the state machine convert to navigation task. The controller send a goal pose topic to navigation package and then the robot drives itself automatically to its goal. After computing the distance and orientation differences between goal and its own position and fining the distance is smaller than a small threshold. The state machine convert to manipulation task state. It will open camera and robot arm. After the camera finding the button object, it will send a 3D position to the arm and finally the arm push the button and reset its pose.

  The main problem of this part is that the localization precision is relative low compared with the precision required from camera. As a result, the robot can not reach a proper position for camera and robot arm. So we have to carefully give initial and goal position. On the other hand, also due to the low precision of localization, the navigation finished signal is difficult to use. Since the robot always fine-tune its position near its goal and waste a lot of time. The worst is the robot still can not reach its goal. So we use distance and orientation differences as our judgment criteria. Once the distance and orientation differences are both smaller enough, the robot will convert to execute the next task. Moreover, since navigation plan its path both in a global and local environments, when the goal position is far away from the robot, its computer can not compute the path in time compared with the map update frequency. To overcome this problem, we first reduce the map update frequency, however the robot will easily go to an obstacle position. So, we decompose the path to a series of positions such that the robot can plan its path in a series of local environments.

## 3.2 Relevant ROS Packages

- **Gmapping**
  The gmapping package provides laser-based SLAM (Simultaneous Localization and Mapping), as a ROS node called *slam_gmapping*. Using *slam_gmapping*, one can create a 2-D occupancy grid map (like a building floorplan) from laser and pose data collected by a mobile robot.
  To use *slam_gmapping*, you need a mobile robot that provides odometry data and is equipped with a horizontally-mounted, fixed, laser range-finder. The *slam_gmapping* node will attempt to transform each incoming scan into the odom (odometry) tf frame.

- **Navigation Stack**
  The Navigation Stack is a ROS package for robotic navigation. It takes in information from odometry, sensor streams and a goal pose and outputs velocity commands to send to mobile base. It can be used for path planning, localization, recovery. In order to use navigation stack, the robot must be running ROS, and have a *tf* transform tree and use the correct ROS Message tyeps. The transform configuration should contains the relationships between each coordinate frames. The sensor information is used to avoid obstacles in the world. The odometry and a *costmap* information is used for planning. At the core of navigation stack is the *move base*. It has a global and local planner where global planner implements some global planner algorithm such as $A*$ algorithm and local planner implements local planning algorithm such as Trajectory Rollout. It also has a global and local costmap which contains information needed for planning. *Recovery behaviors* is also contained such as rotate recovery.

- **SMACH**
  SMACH is a task-level architecture for rapidly creating complex robot behavior. It is useful when you want a robot to execute some complex plan, where all possible states transition can be described explicitly. In other words, you can create state machine by using SMACH. It mainly has following concepts like states, state machine, container, preemption and introspection. "State" can mean different things in different contexts. In SMACH, a state is a local state of execution or equivalently a state corresponds to the system performing some task. This allows the user to focus on what the system is executing and the results from said execution, as opposed to naming the points between execution.
  As its name says, "container" is the container of "state", is contains a dictionary of states and a userdata structure that all of their children can access. It can provide different execution semantics by containing different state, and it has its own ways of specifying transitions for contained states, since transition means different things in different contexts. We can use with keyword or call open() and close() to open a container, then access the opened container by provided function and add the states accordingly.

# 4 System Evaluation

We first evaluate each component of our system respectively.

## 4.1 Navigation

We evaluate the performance of navigation part in two stage. First we control the robot go to goal automatically and find it works well. However, when we

let the robot go to the precise goal, we find it can not go to this place even does not move. Even after reducing cost-map boundary, the robot will sometimes fail to this position because it will first go to this place and then rotate itself. When rotating, the robot often meets collision since the robot is rectangle other than square. Moreover, the precision of the laser scanner is low and the laser scanner is back to the wall, so it more difficult to find the precision position. As a result, the overall successful rate is about 50% even lower.

## 4.2 Button Detection

We implemented three different template-matching strategies,

**Naive Template-Matching:** Directly use template image to perform convolution operation upon input image.

**Hybrid Template-Matching:** Use templates of different buttons to perform convolution upon the input image respectively, and subtract the mean of other activation maps from the objective template's activation map, forming the final activation map.

**Normalized Template-Matching:** Before performing convolution operation upon input image, normalize the template image by subtracting its mean value.

We roughly evaluated the performance of each strategy above from the perspective of localization precision, scale-invariance, lighting condition robustness, and reflection robustness.

Table 1: Evaluation of template-matching strategies.

| strategy | naive | hybrid | normalized |
|---|---|---|---|
| localization precision | good | bad | bad |
| scale-invariance | bad | bad | bad |
| lighting robustness | fine | fine | fine |
| reflection robustness | bad | bad | fine |

## 4.3 Manipulation

Here we evaluate the performance of our button push part in 3 different perspective.
At first, we fix the pose of the car of the arm, and we change the parameter of button detection (template size). In this case, once the detection result is correct, our button push part is almost always success.
After we have found a correct parameter for button detection, we perform the next level experiment. That is, we manually control the car to move to the elevator, and control the pose of the car, then we start our button detection and button pushing. In this case, we found that our button pushing performances

reasonably well. Our button detection can always detect the button. However, the planning of arm will sometimes fails. The overall successful rate is about 70% (4 out of 6 experiments).
At last, we use *SMACH* to do the navigation and perform button detection and pushing. In this case, due to poor precision of navigation, the button will sometimes be out of FOV of the camera and therefore we can not push the button. Even though the button can be detected, planning of our arm sometimes fails. In this case, we haven't push the button successfully out of several experiments.

### 4.4 Overall Evaluation

At last, we perform an overall test. The pipeline of our overall evaluation guided by SMACH is as follows.
First, we load the map and its key points. After checking all modules' states out, we give the robot a goal and it will automatically drive to the goal. However, as is mentioned above, the main challenge of the project is the localization precision. The robot can not arrive the goal position accurately. So after transition from the state of navigation task to detection task, the camera often can not see the button. Also, because of the inaccurate cost-map and laser scanner, the robot always reach a position far away from the goal. As a result, the robot arm can not reach the button even the camera has detected its objective. Due to these reasons, the overall evaluation rarely succeeded. Further tuning of the software and hardware is expected.
Then the robot turns on the camera and start detecting the button. Upon successful detection, it sends a message containing the 3D coordinate to the arm. Finally, based on the measured translation from the arm's finger tip to the button, a feasible trajectory is planned and executed. However, these steps require highly precise localization, so we usually execute the button pushing operation after manual adjustment of the pose of the robot.

## 5 Conclusion

In this paper, we presented the overview of our course project: elevator robot. After discussing related state-of-the-art methods and ROS packages, we discussed the technical details, experiments, and the problems we have met. We evaluated our system in detail to exhibit the advances we have made. To be honest, we did not start from scratch, the system is half-completed, and our task is to combine every part to achieve the final goal. The system, both hardware and software are still far from perfection, problems such as localization precision, button detection robustness, and arm planning rightfulness remain to be solved.

# References


[1] S. Breuers, L. Beyer, U. Rafi, and B. Leibel. Detection-tracking for efficient person analysis: The detta pipeline. pages 48–53, 10 2018. doi: 10.1109/IROS.2018.8594335.

[2] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2):2007–2014, 2019.

[3] F. Del Duchetto, A. Kucukyilmaz, L. Iocchi, and M. Hanheide. Do not make the same mistakes again and again: Learning local recovery policies for navigation from human demonstrations. *IEEE Robotics and Automation Letters*, 3(4):4084–4091, 2018.

[4] P. Ganti and S. Waslander. Network uncertainty informed semantic feature selection for visual slam. In *2019 16th Conference on Computer and Robot Vision (CRV)*, pages 121–128. IEEE, 2019.

[5] M. Görner, R. Haschke, H. Ritter, and J. Zhang. Moveit! task constructor for task-level motion planning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 190–196. IEEE, 2019.

[6] K.-H. Kim, Y. Cheon, S. Hong, B. Roh, and M. Park. Pvanet: Deep but lightweight neural networks for real-time object detection. 08 2016.

[7] R. Lösch, S. Grehl, M. Donner, C. Buhl, and B. Jung. Design of an autonomous robot for mapping, navigation, and manipulation in underground mines. 10 2018. doi: 10.1109/IROS.2018.8594190.

[8] R. Mur-Artal and J. D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[9] S. Patra, K. Gupta, F. Ahmad, C. Arora, and S. Banerjee. Ego-slam: A robust monocular slam for egocentric videos. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 31–40. IEEE, 2019.

[10] Z. Qin, Z. Li, Z. Zhang, Y. Bao, G. Yu, Y. Peng, and J. Sun. Thundernet: Towards real-time generic object detection, 03 2019.

[11] T. Shen, Z. Luo, L. Zhou, H. Deng, R. Zhang, T. Fang, and L. Quan. Beyond photometric loss for self-supervised ego-motion estimation. *arXiv preprint arXiv:1902.09103*, 2019.

[12] X. Wang, Q. Huang, A. Celikyilmaz, J. Gao, D. Shen, Y.-F. Wang, W. Y. Wang, and L. Zhang. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[13] F. Zhu, L. Zhu, and Y. Yang. Sim-real joint reinforcement transfer for 3d indoor navigation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.