A monocular visual odometry pipeline implementation

David Oort Alonso, Patricia Apostol, Rayan Armani, Max Martinez Ruts

***Abstract*— In this work, we present a functional monocular visual odomometry (VO) pipeline implemented in MATLAB using the concepts taught in the *Vision Algorithms for Mobile Robotics* at UZH. The developed VO pipeline provides satisfying local pose estimates and accurate global estimates for a significant number of frames before suffering from scale drift (decay). The authors have achieved these results on 3 diverse and challenging datasets as well as on a custom self-recorded dataset.**

I. Introduction

This report documents the development of a visual odometry (VO) pipeline in the context of the mini-project within the *Vision Algorithms for Mobile Robotics* class. As an additional special feature, a custom video and inertial dataset has been recorded in order to better evaluate the pipeline's performance. The pipeline has been evaluated on the *Parking*, *Kitti* and *Malaga* provided as well as the authors' own dataset. The IMU recorded data was unfortunately not integrated entirely in the pipeline due to time constraints.

The development of the pipeline was composed following three primary goals:

1) A baseline implementation involving a running continuous pipeline.
2) The parallel inertial and video custom dataset recording.
3) Code cleanup and optimisation, including the implementation of trajectory alignment and ground truth comparison.

The report is structured as follows. section II details the methods governing the entire pipeline as well as the authors' own IMU setup, calibration and recording of the custom dataset. The pipeline results´for all datasets and a discussion of these are reported in section III. There are then followed by the main conclusions in section IV.

II. Methods

An overview of the pipeline can be seen in Figure 1. The pipeline has been split into two main parts, namely, the initialisation and continuous operation as shown in the blue and green block respectively. After comparing the performance of the pipeline with self-developed vs. solution code as provided in the course exercises, the solution code was considered overall more consistent and hence employed for a majority of the main functions within the pipeline. An exception has been made for the essential matrix computation `estimateEssentialMatrix` (to replace the equivalent function implemented in the exercises), `vision.pointTracker` (to replace the slower Lukas-Kanade point tracker implemented in the exercises), `triangulate` (to replace the slower linearTriangulation implemented in the exercises) and `estiateWorldCameraPose` (to replace P3P RANSAC implemented in the exercises). These have been implemented using prebuilt Matlab functions.

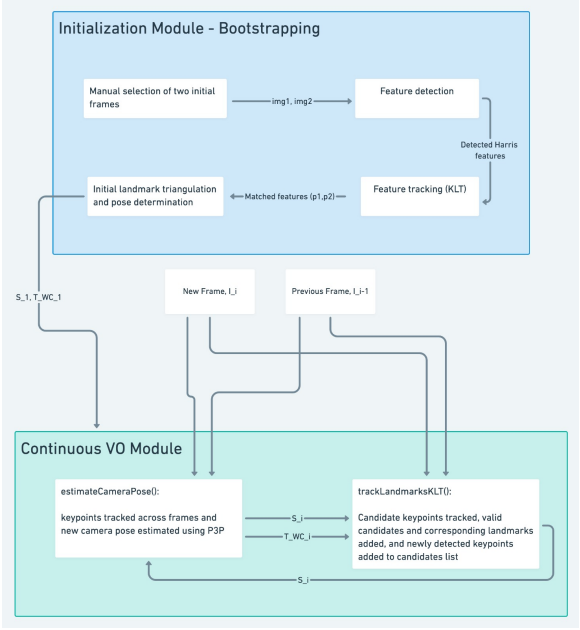

Fig. 1. Final VO pipeline

As an additional feature, an own-dataset has been recorded in the Student Project House at ETH Zentrum. The dataset relies on both a video recording as well as IMU data. The methods for the primary pipeline in Matlab as well as a detailed account of the Camera-IMU setup, calibration, and data recording are detailed in the subsections below.

A. Initialisation

The initialisation block is split into two primary parts. The first involves the loading of the corresponding datasets as well as their respective tuning parameters from the configuration files. The second performs the initial bootstrapping in order to determine the world camera pose and triangulate a first set of landmarks. The overall steps are listed as follows and key aspects are elaborated on subsequently:

- Loading of the dataset and extraction of the ground truth data.
- Selection of the bootstrapping frames.
- Loading of the tuning parameters from the configuration - `dataset.json`.

- Detecting an initial keypoint set on the first bootstraping frame using Harris corner detection - `detectKeypoints`.
- Using the KLT tracker to track the initial keypoint set on the second bootstrapping frames - `vision.PointTracker`.
- Estimating the relative camera pose between the two bootstrapping frames and triangulating the 3D landmarks and - `structureFromMotion`.

1) Frame selection and configuration file: Configuration files for each datasets of the form `'dataset.json'` were made in order to allow for individual tuning as well as better structuring of the dataset parameters. When initialising the pipeline, the first and third frame of the dataset were selected. This was found to be a good balance to have enough keypoint correspondences and a reasonable baseline to facilitate triangulation. The initial set of keypoints is determined using the Harris corner detector as implemented in the exercises.

2) Feature detection and tracking: As mentioned previously, the Harris corner detector was ultimately used in the final version of the pipeline. In the early stages, the FAST and SURF keypoint detectors were also experimented with using the prebuilt Matlab functions. The FAST descriptor however behaved rather poorly in the number of keypoints initially detected, which resulted in overall lower performance when triangulating new landmarks as well feature tracking. Meanwhile, although the SURF detector initially detected a large amount of keypoints, performance when integrating with KLT remained poor. Hence, in the scope of this application, the Harris corner detector was settled upon due to its computational efficiency and as well as higher localization accuracy and hence better integration with the KLT tracker.

The tuning parameters for the Harris corner detector are the corner patch size, the threshold parameter and the non-maximum suppression radius. Additionally, the number of initial keypoints was also adjusted accordingly. The final values of the tuning parameters as well as the initial number of keypoints found most optimal together with a justification are reported within the results and discussion detailed in the section III.

The first approach implemented for feature tracking was to apply feature matching using patch descriptors from the obtained obtained Harris features. However, the amount of unmatched or incorrectly matched features was found to be considerably high, thereby severely increasing the computation time in RANSAC methods and generally decreasing the performance of the pipeline. The second approach was to implement KLT using the code implemented in the exercises. Nevertheless, the nature of the implementation resulted in a very large computation time that made impractical. Thus, we opted to use a Matlab built-in class `vision.PointTracker` that runs much faster and also implements pyramid images. We observed that with this implementation, the amount of successfully tracked keypoints increased significantly.

3) Landmark triangulation and pose estimation: As a final step in the initialisation process, structure from motion was performed on the final matched points with the aim of determining the initial camera pose as well as triangulating a first set of landmarks. The matched keypoints in image coordinates together with the camera intrinsics were passed within `estimateEssentialMatrix` which computes the essential matrix using MSAC, a variant of the RANSAC algorithm, which directly optimises for the inliers when computing the essential matrix. As the camera was already calibrated, there was no need for computing the fundamental matrix.

Given the essential matrix, the rotation and translation of the camera with respect to the world frame were extracted using `disambiguateRelativePose`. These were then used to compute the 'M' matrices used in the Matlab `triangulate` method in order to get an initial set of 3D landmarks. The effects of triangulation on the resulting pipeline and how these were addressed are further elaborated on in subsection II-B as well as as section III.

B. Continuous operation

The continuous operation component of the pipeline relied on two main functions. The first being `estimateCameraPose` and the second being `trackLandmarksKLT`. In order to run the pipeline in a continuous setting, as recommended in the project description, each keyframe was associated with a particular state S^i of the following form:

$$S^i = (P^i, X^i, C^i, F^i, T^i) \tag{1}$$

P^i refers to the Harris detected keypoints in each frame whilst X^i represent the 3D landmarks associated to the detected keypoints. This distinction was required due to the fact that not every keypoint will be associated to a triangulated landmark. The keypoints which are not associated are then removed. In order to triangulate new landmarks to compensate for the lost tracked landmarks over time, the state S^i is also also associated with three other attributes - C^i represents a set of candidate keypoints to be tracked into a subsequent frame, F^i holds the first observation of these keypoints, and T^i the camera poses at the first observation. The continuous update of this state relied on two primary functions `estimateCameraPose` and `trackLandmarksKLT` is presented in Figure 1.

1) Frame processing and pose update: The inputs and outputs of `estimateCameraPose` are defined as follows:

```
function [S_i, T_WC_i] =
    estimateCameraPose(img_i,
    img_prev, S_prev, args)
```

As input `estimateCameraPose` considers both the incoming and previous keyframes as well as the past state. The function stores the the set of keypoints and associated landmarks from the previous frame. Subsequently, the `vision.PointTracker` is initialised and tracks the set

of keypoints (P) and landmarks (X) in the new image frame. Given these, the camera pose is determined using the prebuilt `estimateWorldCameraPose` method from the 3D - 2D correspondences. Again, this Matlab function was chosen due to its direct incorporation of the RANSAC variant, MSAC in the P3P algorithm. The output then updates the P and X state attributes exclusively, whilst all others remain the same.

2) Landmark update and triangulation: This updated current state (S^i) is then passed as input to `trackLandmarksKLT` whose outputs and inputs are defined as follows:

```
function S_i = trackLandmarksKLT(S, images, T_WC_i, args, beta)
```

To be noted here is that the input S directly refers to the output of `estimateCameraPose` for the current state S^i. `trackLandmarksKLT` hence aims at updating the state S^i by tracking a set of candidate keypoints and triangulating a set of new landmarks whilst discarding those no longer tracked as well as duplicates. As noted in the function definition, the estimated pose serves entirely as an input to the function and unlike S^i, is never updated within.

Given the state S^i, a set of candidate keypoints is taken from the previous frame and tracked across the next frame using the `vision.PointTracker`. The successfully tracked points are then stored in a new list of candidates each time the function is called. Given the successfully tracked candidates, the points' first observation and pose array are also updated using the `pointvalidity` output of the `pointTracker` object.

The tracked candidates keypoints are then used to triangulate a new set of landmarks using the stored pose at the first observation (T_{WC}^{first}) as well as the current image pose (T_{WC}^{i}) passed into the `trackLandmarksKLT` as an output of `estimateCameraPose`.

Very importantly, a few issues were encountered during the triangulation process which decreased the quality of the estimation. The first was related to the uncertainty in the triangulated 3D point. The smaller the angle between the two keyframes' visual rays, the greater the uncertainty in the landmark's position. This aspect was addressed as suggested in the project description. The angle α between the camera bearing vectors was defined and compared to a threshold chosen after tuning. Any triangulated landmarks for which α was greater than the threshold were considered suitable (thereby transferred from the candidate pool to the accepted pool). The rest were kept in the candidate pool).

Within the monocular VO, another issue was related to the large scale drift present after about 500-1000 frames, depending on the dataset. In long datasets, like the *Malaga*, *Kitti* and our custom dataset, it can be clearly observed. It was noted that landmarks which were too far away (and which were not already discarded using the previous angle heuristic) gave the impression that the camera frame was not moving at all. This is because the accuracy of keypoint detection and tracking is comparable to the actual motion of the far away landmark in image space. This leads to some keypoints not moving at all which lead to an underestimation of the camera motion, which in turn leads to a smaller distance of newly triangulated landmarks, which leads to an ever decreasing scale for the map and trajectory of the camera. Unfortunately however, adding the condition for new landmarks to be closer than a certain distance did not seem to work particularly well, and in fact it deteriorated the quality of the triangulated landmarks. This may be due to that fact that the parameters were tuned for a different distribution of keypoints. It became too time consuming to find parameters working better with this new heuristic. Hence, the angle heuristic described previously was the one settled upon in the final pipeline.

Finally, the Harris corner detector was run on the new frame in order to detect new candidate keypoints. These were then appended to the candidate keypoints state attribute, and the finalised state S^i was passed on again as a previous state into the `estimateCameraPose` function in order to estimate the following camera pose.

C. Trajectory alignment and bundle adjustment

In order to better compare the data to the ground truth, trajectory alignment was introduced exactly as specified in *Exercise 9* at the end of the trajectory. The alignment was implemented in the `alignEstimateToGroundTruth` function. Nevertheless, the alignment does not solve the scale drift problem. For this, bundle adjustment was considered. Unfortunately, due to time constraints, there was only time to implement bundle adjustment for initialization but the approach for implementing it in a sliding-window approach was discussed, whereby the Jacobian would be modified (the derivative of the error with respect to the first few poses being set to 0) so as to prevent the optimization routine to modify the starting poses and thereby guarantee smoothness across windows.

D. Validation: Visual inertial odometry extension

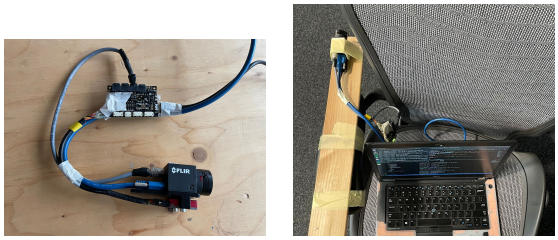

Fig. 2. Dataset Acquisition Hardware and setup

1) IMU and video data collection: The custom dataset was recorded using a *VN100 IMU* and a *Blackfly S USB3* camera, with the thought of using it to develop a Visual-Inertial odometry pipeline. We used the open source Versavis framework [2] to record synchronised camera frames and IMU measurements at 20Hz and 200Hz respectively. The IMU and camera were secured to a 3D printed mount to keep the camera-imu extrinsics stable, and the assembly was

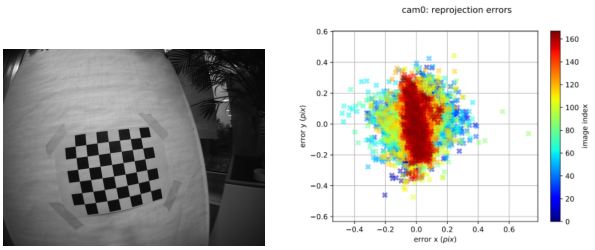

Fig. 3. Calibration setup

then attached to a wheeled chair (figure 2). The chair was then pushed around to complete loops around an office space. The data stream was recorded as a rosbag. To calibrate the setup, the Kalibr toolbox [3] was used on a recording of a checkerboard pattern to obtain the camera intrinsics and the camera-IMU extrinsics. Figure 3 shows the re-projection error after the camera was calibrated, with more variance in the y direction. The IMU Noise model required by the Kalibr toolbox for the VN100 was calculated following the repository's wiki [4] using values from the IMU's datasheet [5]. The camera being equipped with a fish-eye lens, the open-source tool "image undistort" was used to correct lens distortion, as it was straightforward to use with the rosbag and calibration outputs formats. The resulting rosbag containing IMU, undistorted camera images and modified intrinsics was then unpacked with a custom python script to extract a .csv file with IMU measurements and numbered images in .png format.

2) Integration into baseline pipeline:

III. Results and Discussion

The following section details the final output trajectories for each dataset and areas of interestand improvement are elaborated on. Additionally, the screencasts provided next to this report have been recorded on a machine with specs summarised in Table I.

Operating system	Ubuntu 20.04
Cores	12
CPU frequency	2.6GHz
RAM	16GB
	No parallelisation, single thread performance

TABLE I

Machine specifications

A. KITTI Dataset

The results for the Kitti dataset show good performance of the pipeline in pose estimation and landmark triangulation relative to the recent frames. However, the global trajectory suffers from scale drift to an extent that the last frames end up being very close to each other. For this reason, we present only the first 850 frames in figure Figure 4. After this point, the trajectory alignment would not allow for a clear comparison to the ground truth data. Nevertheless for reference, this issue in the entire trajectory is clearly visible in the project provided screencast. The accumulated scale drift could ideally only be addressed through the successful implementation of bundle adjustment in the continuous setting.

Tuning parameter	**Value**
Harris threshold parameter κ	0.04
Harris corner patch size	15
Harris non-maximum suppression radius	11
Maximum number of keypoints	1000

TABLE II

Tuning parameters for the *Malaga* and *Kitti* datasets

Tuning parameter	**Value**
Harris threshold parameter κ	0.1
Harris corner patch size	15
Harris non-maximum suppression radius	11
Maximum number of keypoints	1000

TABLE III

Tuning parameters for the parking and custom dataset

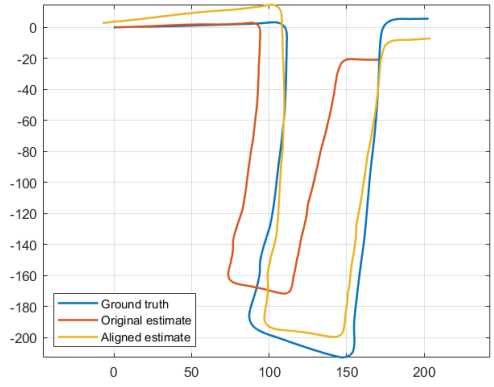

Fig. 4. Aligned *Kitti* dataset trajectory after the first 850 frames

The tuning parameters for the *Kitti* dataset are summarised in Table II.

B. Parking Dataset

The final parking dataset trajectory is shown inFigure 5. The pipeline shows great performance after alignment across all frames and very minimal scale drift is present. This may be attributed to the 'simplicity' in the dataset, minimal rotation, and movement perpendicular to scene (hence more accurate pose estimation). For this dataset exclusively, bundle adjustment may hence not be necessarily required.

The tuning parameters for the parking dataset are summarised in Table III.

Increasing the threshold parameter κ to 0.1 (in comparison to 0.04 for the other provided datasets) and thus also the corner 'sensitivity', resulted in a significant trajectory improvement by decreasing the number of error inducing corners detected. This was not found to be beneficial in other datasets

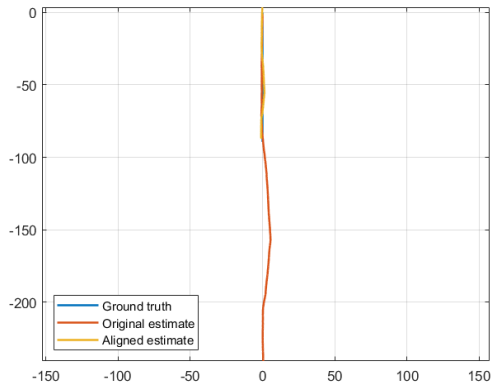

Fig. 5. Aligned *Parking* dataset trajectory across all frames

because it would reduce the amount of detected keypoints). Altering the other feature detection tuning parameters was found to have no significant effect.

C. Malaga Dataset

Given that no ground truth data was provided, solely the estimation plot has been depicted in Figure 6.

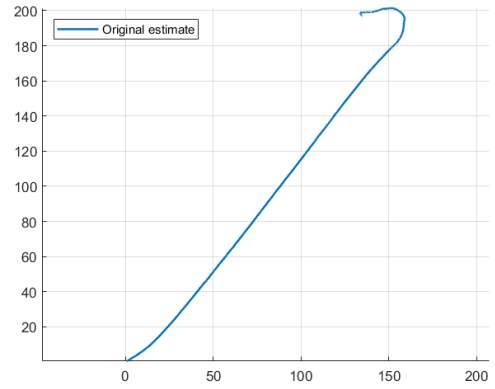

Fig. 6. Final *Malaga* dataset trajectory across all frames

The final tuning parameters for the *Malaga* dataset remained exactly the same as those for *Kitti* and are hence summarised in Table II.

Overall, the pipeline seems to perform locally well on the *Malaga* dataset. Nevertheless as expected, the dataset suffers from very significant scale drift, also clearly visible by the minimally recognisable change in trajectory after 1000 frames (particularly visible in the screencast). Extracting the ground truth from the GPS data and aligning the output would result in a more accurate comparison. Finally of course here again, a successful implementation of bundle adjustment in the continuous setting could help reduce the overall drift.

D. Custom Dataset

The visual odometry pipeline was run on the collected dataset ignoring the IMU measurements. The main parameters tuned were the κ coefficient for the Harris cornerness function- which ended up the same as for the *Parking* dataset and summarised in Table III. Results are shown in figure Figure 7.

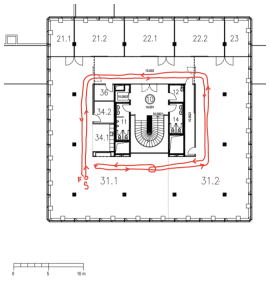

Fig. 7. Approximate true motion, as no accurate ground truth was collected

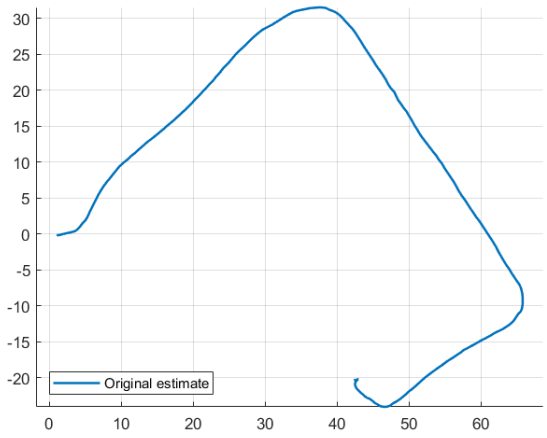

Fig. 8. Custom dataset trajectory across the first 400 frames

On the custom dataset, scale drift in the pipeline is very apparent: from figure 8 we can see that the overall trajectory estimated by the pipeline is gradually shrinking in scale. Several factors could explain the shortcomings of the pipeline of this dataset, the first being that the speed of the turn combined with the low frame rate introduces a very large displacement between matching features as well as motion blur. A second reason concerns the sharp changes in illumination that happen at the pipeline's failure points, as the camera goes from a darker area of the room to pointing directly at the windows on a bright day. Solutions to these issues could be changing the camera parameters such as gamma and exposure to output a less contrasted and more texture rich image, as well as sampling at a higher frame rate to minimise the displacement between landmarks in consecutive images during faster movements. Another way to improve performance would be to leverage the IMU data, which could be especially useful to combat scale drift as it anchors the trajectory estimation to an absolute scale. A more quantitative evaluation of the pipeline performance would need ground truth data. In the absence of GPS signal indoors and lack of precise motion capture systems, a ground truth

dataset could be generated using a verified Visual Inertial Odometry framework, such as ROVIO [7]. A video showing ROVIO running on our custom dataset can be found here. Unfortunately, we could not tune ROVIO in time for this project and the out-of-the-box performance was not good enough to be used as ground truth.

IV. Conclusions

On the given datasets, the pipeline achieved very good local results across the entire trajectory as well as good global performance for a number of frames. Nevertheless, the more complex datasets did suffer from significant scale drift. At the same time, a custom dataset was successfully recorded and calibrated using the Kalibr toolbox. Athough the pipeline failed towards the end of the custom dataset when encountering a more tricky setting, options to address this failure have been presented in section III. Given more time, the main areas of improvement for the current pipeline are summarised below:

- The successful implementation of bundle adjustment in the continuous setting in order to achieve better global performance and combat decay.
- An integration with ROVIO extracting the ground truth for the custom dataset in order to better evaluate its performance.
- A successful integration of the IMU data, addressing the authors' original goal of building a visual inertial odometry pipeline, in order to better combat the scale drift issues and achieve excellent global performance across the entire trajectory.

References

[1] M. H. Mohd, N. A. Azman, and N. A. Aziz. Alternative Stable States and Limit Cycles in a Three-Species Ecological System, AIP Conference Proceedings 2266, 2020.

[2] F. Tschopp, M. Riner, M. Fehr, L. Bernreiter, F. Furrer, T. Novkovic, A. Pfrunder, C. Cadena, R. Siegwart, and J. Nieto, "VersaVIS—An Open Versatile Multi-Camera Visual-Inertial Sensor Suite," Sensors, vol. 20, no. 5, p. 1439, Mar. 2020 [Online]. Available: http://dx.doi.org/10.3390/s20051439

[3] Furgale, P., Timo Hinzmann, T., Oth, L., Schneider, T., Rehder, J. and Maye, J., 2018. Kalibr · ethz-asl/kalibr. [online] GitHub. Available at ¡https://github.com/ethz-asl/kalibr¿

[4] Furgale, P., Timo Hinzmann, T., Oth, L., Schneider, T., Rehder, J. and Maye, J., 2018. IMU Noise Model · ethz-asl/kalibr Wiki. [online] GitHub. Available at: ¡https://github.com/ethz-asl/kalibr/wiki/IMU-Noise-Model¿

[5] Vectornav, 2014. VN100 User Manual. [online] Vectornav. Available at: ¡https://www.eol.ucar.edu/system/files/VN100manual.pdf¿

[6] Autonomous Systems Lab, ETHZ, 2018. GitHub - ethz-asl/image: A compact package for undistorting images directly from kalibr calibration files. Can also perform dense stereo estimation. [online] GitHub. Available at: ¡https://github.com/ethz-asl/image_undistort¿

[7] M. Bloesch, S. Omari, M. Hutter and R. Siegwart, "Robust visual inertial odometry using a direct EKF-based approach," 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015, pp. 298-304, doi: 10.1109/IROS.2015.7353389.