TrimBot2020 Deliverable D2.2

Manipulator and tools version 1, running Open-Loop Motion Planning

Principal Author:	Wageningen University (WU) & Wageningen Research (WR)
Contributors:	Albert-Ludwigs-Universität Freiburg (ALUF)
Dissemination:	CO

Abstract. Deliverable 2.2 is a hardware and software deliverable, in which we demonstrate open-loop trimming of boxwood bushes with a stationary robotic manipulator. It shows the successful integration of vision pipeline for the cameras on the arm and the motion planning pipeline for the manipulator. This report serves as a documentation overview of the deliverable and comes along with a video clip showing the robot in action. Note that Deliverable 2.1 contains background information on the components presented here.

Deliverable due: Month 18

Contents

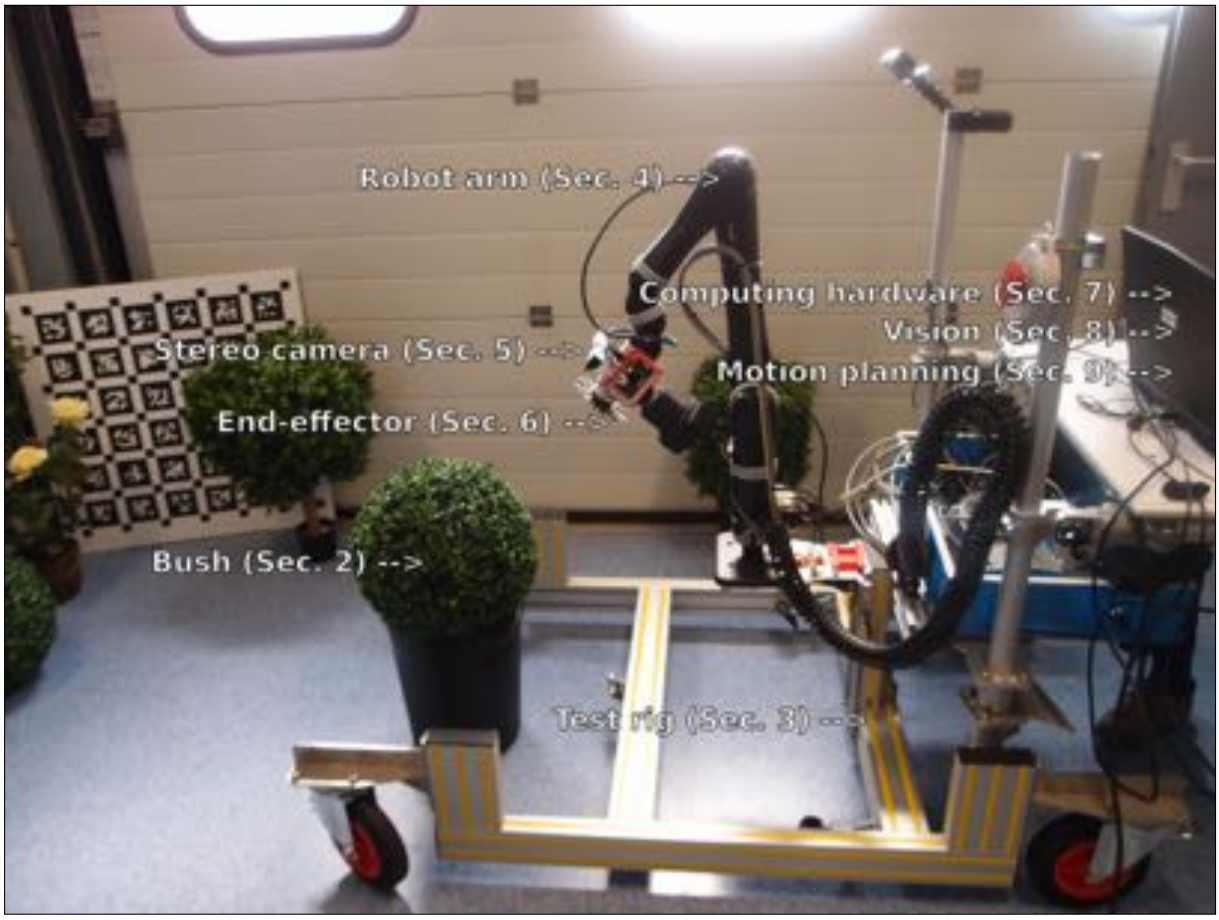

Figure 1: Demo setup for Deliverable 2.2: the world's first autonomous boxwood trimming robot. Each component is explained in a separate section below. Note that this image shows an artificial bush, instead of the real one that can be seen in the accompanying video.

1 Introduction

TrimBot2020's "manipulator and tools version 1, running open-loop motion planning", or: the world's first autonomous boxwood trimming robot, is displayed in Figure 1. This stationary robot serves as a stepping stone towards TrimBot2020's mobile trimming robot prototype. While other TrimBot2020 partners develop the garden mapping and navigation functionality, this deliverable focuses on the trimming action itself.

Being a hardware and software deliverable, Deliverable 2.2 consists of:

- this report,
- a video showing the trimming robot in action[1],
- a spherical boxwood bush (Section 2),
- a test rig that can be moved manually (Section 3),
- a robot arm (Section 4),
- a stereo camera (Section 5),
- a custom end-effector (Section 6),
- computing hardware (Section 7),
- a vision algorithm with implementation (Section 8), and
- a motion planning algorithm with implementation (Section 9),
- a motion controller with implementation (Section 10);

2 Bush

We used a spherical boxwood bush, with a diameter of 40cm. The goal was to trim it back a few centimeters, while maintaining its spherical shape. The artificial bush in Figure 1 was replaced by a real one later, as can be seen in the accompanying video[1].

3 Test rig

A custom test rig was designed and built for Deliverable 2.2: a stationary platform that can be moved manually if needed. It contains a sturdy mount for the robot arm, as well as a laptop stand for easy developer access (see Figure 1).

The test rig differs from the real vehicle that is going to be used in TrimBot2020 in a few aspects:

- The real vehicle introduces more areas in the workspace of the robot that cannot be accessed due to collision. This, in turns, means that the reachable area of the bush from a single vehicle pose will be smaller.

[1] https://gitlab.inf.ed.ac.uk/TrimBot2020/General/tree/master/deliverables/D2.2/video/trimbot_d2-2.mp4

- In the real vehicle the base of the arm is at a lower height with respect to the ground compared to the test rig configuration. Anyway it is worth to notice that in our demo the bush was inside a pot, whereas in a garden test the bush would be on the ground.
- The test rig is likely to be more stable than the final vehicle (in this regard tests need to be performed when the vehicle final prototype is available).

4 Robot arm

A Kinova Jaco[2] robot arm was mounted on the test rig. This is the same arm that will be used for the final TrimBot2020 mobile prototype. The arm's 7 rotational joints allow for the dexterous movement that is required to perform the trimming task. The Kinova Jaco[2] is light-weight, does not need a contol box, and it consumes little power. This makes it a suitable choice for TrimBot2020, given the mobility requirements, and the tasks that it has to perform. This choice is further motivated in Deliverable 2.1.

5 Stereo camera

In order to perceive the boxwood bush, the robot uses a stereo camera system, developed at ETH Zürich. The two cameras had a diagonal FOV of 68°. They were placed 10cm apart on an aluminum mount, with an offset of 6cm above the end-effector frame (see Figure 3). They were calibrated intrinsically and extrinsically using the Kalibr Toolbox [2]. In Figure 2 there are some examples of left camera image, right camera image and disparity map of the bush with the given setup.

6 End-effector

Several cutting principles may be feasible for trimming boxwood. In Deliverable 2.1, we showed that circular saw blades, spinning in opposite directions, are the best option for TrimBot2020. Based on this principle, we designed and built a custom end-effector, using long saw teeth of a particular shape, that can easily catch the boxwood branches between them. The end-effector and stereo camera are shown in Figure 3.

7 Computing hardware

We used a Schenker XMG U705 laptop[2] to run all software modules for Deliverable 2.2: computer vision, motion planning, and miscellaneous components. The laptop specs are as follows:

[2] http://www.xmg.gg/u705/?LANG=EN

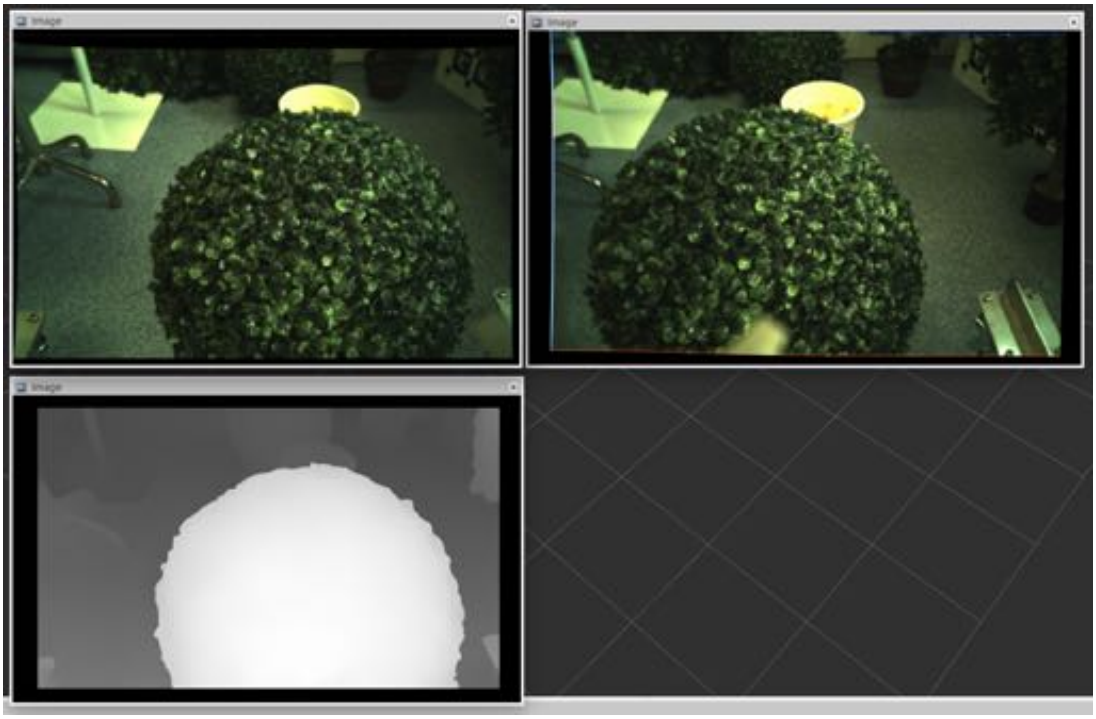

Figure 2: Top row, left corner: left camera image. Top row, right corner: right camera image. Bottom row, left corner: disparity map.

- NVIDIA GeForce GTX 980M 8GB
- Intel Core i7-4790T (4 cores, 8 threads)
- 32GB DDR3-1600 (4x8GB)
- 1TB SATA-III SSD (Samsung 850 EVO)

The modules communicate with each other through ROS (Robot Operating System). A similar gaming laptop (with higher specifications) will be mounted on the final, mobile TrimBot2020 prototype. This laptop will run the computer vision algorithms, while motion planning (among other things) may run on a Pokini i2 computer[3] in the future.

8 Vision

We refer to the Vision part as those software modules which take the visual input from the robot's arm cameras and produce the data needed by the Motion planning modules. Specifically, in the context of this Deliverable this comprises

1. rectifying and undistorting the raw camera images,

[3] http://www.pokini.de/de/produkte/fanless-pcs/pokini-i2

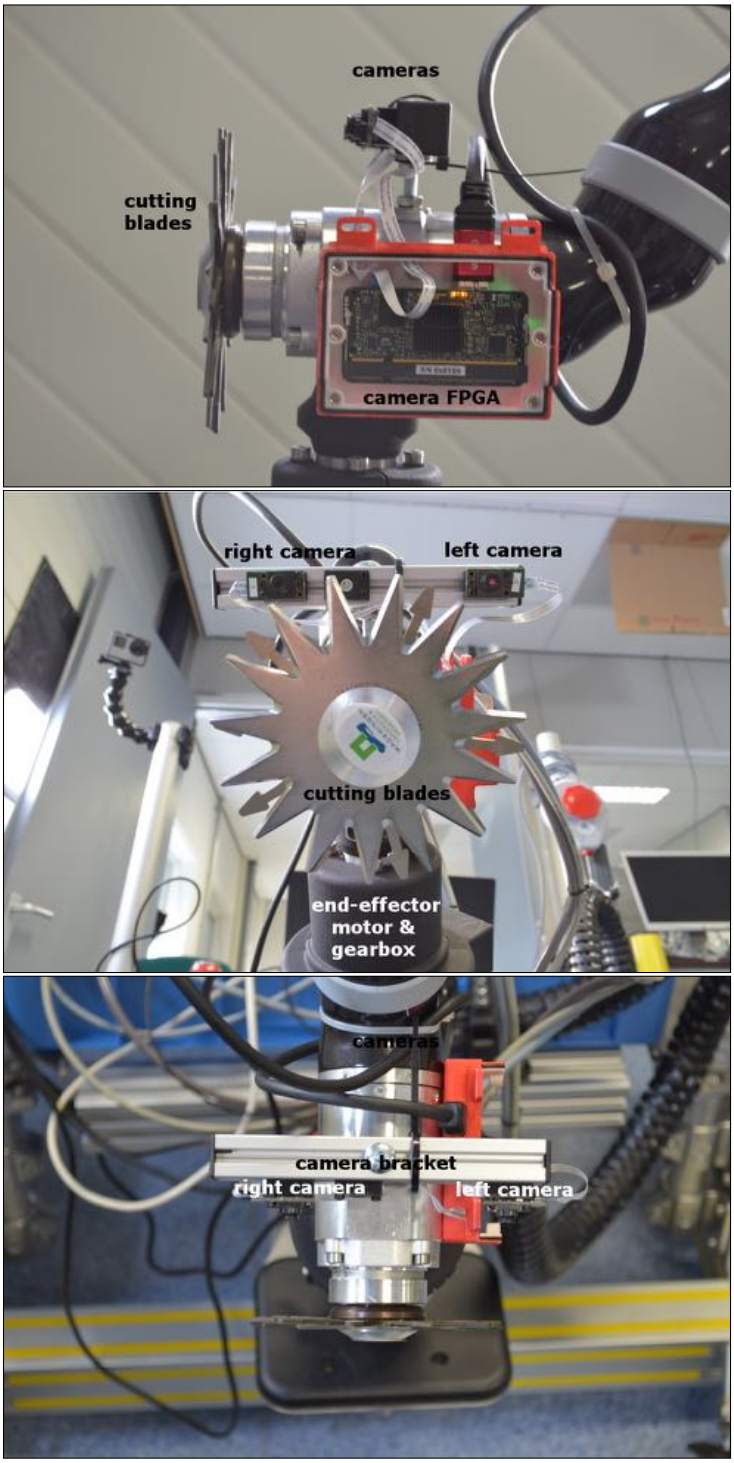

Figure 3: Custom end-effector with stereo camera. Side view (top image), front view (middle image) and top view (bottom image)

2. feeding the undistorted images through the DispNet to obtain a disparity map (see Figure 4),
3. reconstructing a 3D pointcloud of the observed scene, and
4. fitting a sphere model into the pointcloud where the spherical bush is believed to be visible.

We use the Kalibr [2] framework to calibrate the camera intrinsics and stereo-extrinsics. The raw images are then rectified and undistorted before being fed into the DispNet. Using the known camera intrinsics, the resulting disparity map is then used to reconstruct a metric 3D pointcloud (in camera space).

For more details about the camera setup and disparity estimation, we refer the reader to the document of the Deliverable D5.1.

To fit a sphere model, we first try to obtain a most likely position of the bush in the camera image. We use a naive approach and simply use the pixel position with the largest disparity value as the starting position (this could be made more robust by also including semantic segmentation information, but the respective modules were not yet available at the time of writing, and our approach proved to be sufficient for this Deliverable).

Then, we extract a partial pointcloud by sampling from a disk surrounding the estimated bush center in the disparity map. We then use a direct least-squares algorithm [1] to obtain the position and radius of the sphere which best describes this point cloud (see Figure 5). Note that the disparity map has invalid regions close to the image borders because the input images must be cropped to remove undistortion artifacts which are not handled by the DispNet. Due to inaccuracies and estimation errors in the disparity map, the initial position likely does not describe the actual center of the bush, and so we perform multiple iterations of the direct algorithm (in each iteration, a new partial pointcloud is sampled from the reprojected estimated sphere position of the previous iteration).

It is worth to remark that from a single viewpoint the visible portion of the bush surface is less than 50% of the total surface, which has repercussions on the quality of the mesh fitting. In the future, the system will be improved in order to exploit the mesh fitting computed from multiple camera viewpoints. This will enhance the accuracy of the acquired position and size of the bush.

Communication between the vision pipeline's modules (and towards the following motion planning) is done via ROS messages (for images we use `sensor_msgs::Image`, disparity maps are `stereo_msgs::DisparityImage`, and the sphere mesh is a `shape_msgs::Mesh`).

9 Motion planning

The motion planning algorithm for bush trimming is divided into three submodules: 1) the planning setup module, 2) the coverage planning module, 3) the trajectory planning module. The

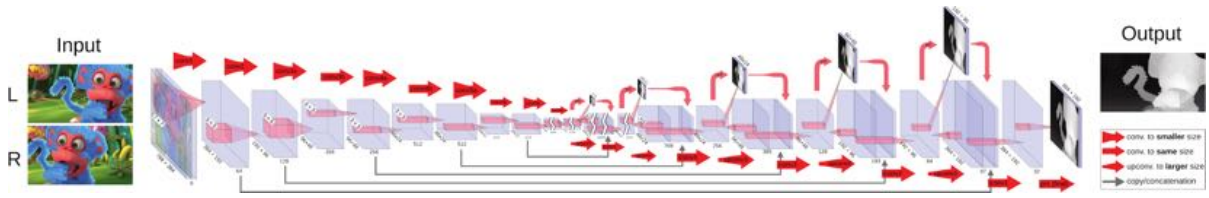

Figure 4: Disparity estimation. We use the DispNet [3], a state-of-the-art neural network algorithm, to estimate a disparity map for rectified stereo images provided by the arm cameras.

planning setup module processes the input bush target shape to generate the search space for the actual planning task. The coverage module produces a list of intermediate joint configurations allowing to achieve coverage of the cutting area with the tool (see Figure 7). The trajectory module interpolates the intermediate joint configurations into a smooth reference trajectory. An overview of the motion planning algorithm is shown in Figure 6. In the Figures it is possible to observe how limited is the swept percentage of the target bush boundary when the trimming path is calculated from a single base pose. It is easy to spot a triangular dead zone, caused by a forbidden angular range of the second joint of the arm due to self-collision.

The motion planning algorithm was implemented in Matlab R2017a and V-REP[4] 3.3.2. Communication between V-REP and Matlab was achieved through the Matlab external API for V-REP. V-REP inverse kinematics solver was used to generate the candidate joint configurations for the coverage planning. The V-REP scene setup is shown in Figure 8. Robotics Toolbox[5] was used to connect to a ROS network from Matlab.

The motion planning algorithm reads a `shape_msgs::Mesh` ROS message encoding the target bush shape extracted by the vision algorithm. The output trajectory is encoded into a `trajectory_msgs::JointTrajectory` ROS message.

10 Motion control

The `wpi_jaco_wrapper`[6] driver was used to interface with the robot arm. This driver is a ROS wrapper of Kinova API[7], and allows to set a JointTrajectory as a ROS action goal. In order to track the reference trajectory, the driver uses a PD control to compute instantaneous angular speed commands to the joints. The joint position errors are computed by performing a readout on the joint encoders. The driver has been modified to allow decoupling between the encoders readout frequency and the joint speeds command frequency.

In the video we recorded the trimming tool seems to follow the curve of the bush. It is worth to remark that in a stationary setup approximately $\frac{1}{4}$ of the bush surface is actually swept by the trimming tool. In the future, order to perform full coverage in a stationary setup, the bush will be rotated and a trimming trajectory will be executed for each untrimmed portion. A rigorous evaluation of the trimming quality will be performed in D2.3.

[4] http://www.coppeliarobotics.com/
[5] https://it.mathworks.com/products/robotics.html
[6] http://wiki.ros.org/wpi_jaco_wrapper
[7] http://www.kinovarobotics.com/innovation-robotics/products/software/

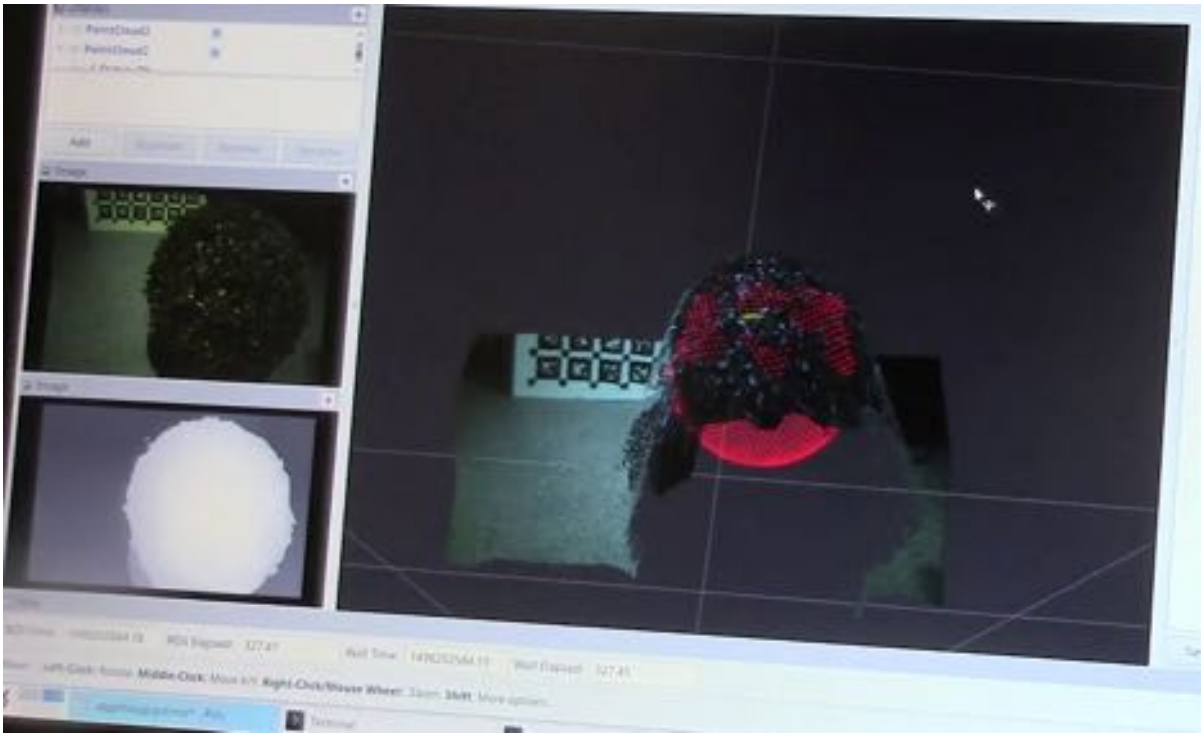

Figure 5: Vision pipeline. This screenshot of the ROS application ”RViz” visualizes all the components of the current vision pipeline: On the left, the stereo camera’s left image (rectified and undistorted) and the corresponding disparity map estimated by the DispNet (see Figure 4). On the right, the textured pointcloud reconstructed from the disparity map (using the known camera intrinsics) and the input image, and the least-squares fitted sphere model (shown as a red poincloud) which tries to describe the underlying shape of the bush are visualized.

11 Conclusion

We conclude that Deliverable 2.2 is a success. The video shows that the robot can successfully trim back spherical boxwood. For Deliverable 2.3, and the planned corresponding journal paper, we will improve upon this first proof-of-principle, and perform a quantitave performance evaluation. The report for Deliverable 2.3, “evaluation and dissemination of manipulator and tools version 1 plus open-loop motion planning”, is due in December 2017.

The D2.2 demonstration video can be downloaded at the following link:
https://gitlab.inf.ed.ac.uk/TrimBot2020/General/tree/master/deliverables/D2.2/video/trimbot_d2-2.mp4

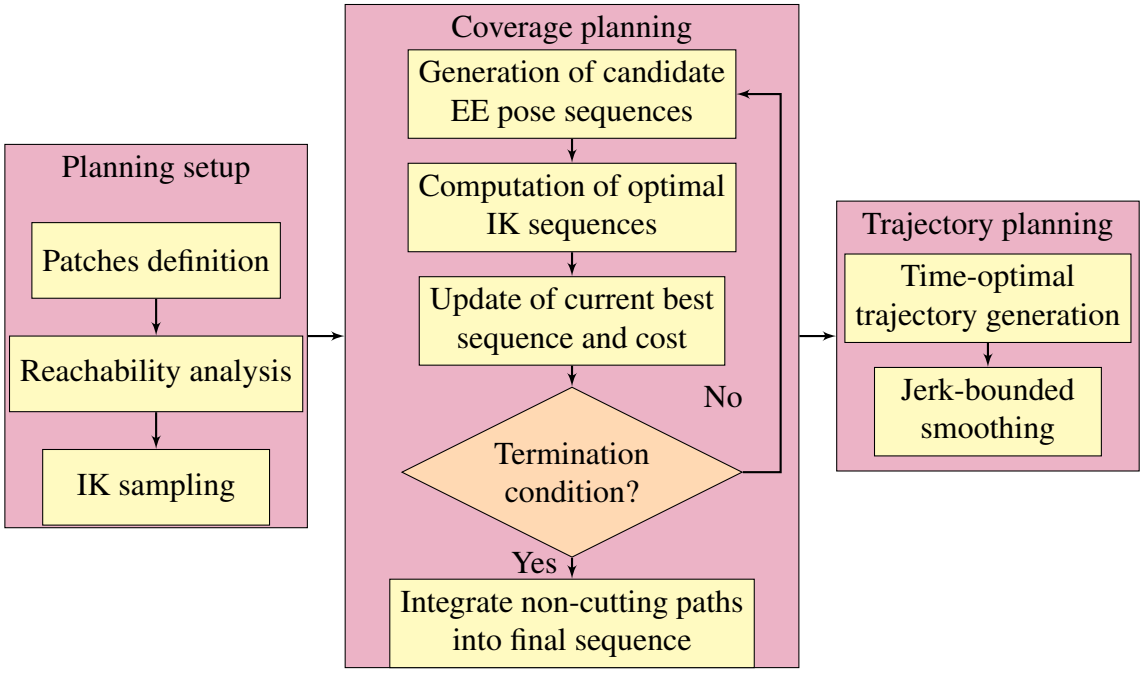

Figure 6: Coverage trajectory planning algorithm for bush trimming. In the diagram IK = Inverse Kinematics, EE = End-Effector.

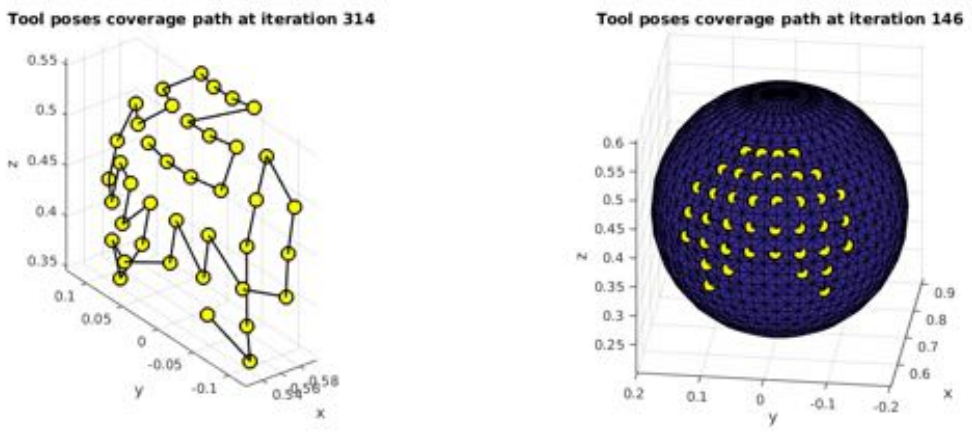

Figure 7: Left: Example of computed tool coverage plan. Right: Example of computed tool coverage plan, visualized on top of the target bush boundary.

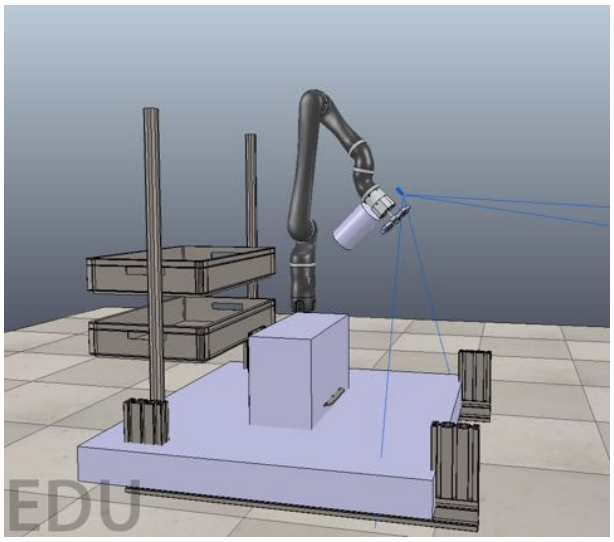

Figure 8: V-REP scene for motion planning.

References

[1] David Eberly. Least squares fitting of data. *Chapel Hill, NC: Magic Software*, 2000.

[2] Jérôme Maye, Paul Furgale, and Roland Siegwart. Self-supervised calibration for robotic systems. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 473–480. IEEE, 2013.

[3] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. arXiv:1512.02134.

