TrimBot2020 Deliverable D2.4

Manipulator and tools V2, Closed-Loop Motion Planning

Principal Author:	Wageningen University (WU) & Wageningen Research (WR)
Contributors:	A.-Ludwigs-Univ. Freiburg (ALUF)
	University of Edinburgh (UEDIN)
Dissemination:	CO

Abstract. This deliverable describes the revised design of the end-effectors for bush trimming and rose cutting, as well as the vision modules and control components for closed-loop bush trimming and rose cutting. The motor that drives the revised bush trimmer tool is mounted on the 4th axis of the robot arm and will be connected by a flexible shaft to the tool. This results in a much lighter and smaller device. The length of the rose cutter tool is reduced as well. For both end-effectors 3d printed housings have been designed that allow a rigid mounting of the tool cameras. Current accuracy issues in controlling the arm have been investigated and have revealed that no mechanical deformation occurs when applying a load to the arm. A difference between the commanded and actual poses of the arm was however observed. To increase the accuracy of the arm control calibration markers will be placed on the end-effectors that allow visual arm tracking and pose estimation. The vision module for bush trimming will use the tool-mounted stereo cameras for depth sensing and target shape fitting using multiple camera poses. The vision module rose cutting will use the tool-mounted stereo cameras for CNN based branch segmentation and bud eye detection. From the full list of bud eye locations the desired cutting position are selected. The closed loop trajectory control for bush trimming will apply the concept of trajectory adaptation based on the observed trimming score. The trajectory control for closed loop rose cutting will apply a reinforcement learning algorithm as a visual servoing controller. This visual servo control algorithm will process real-time images with the goal to minimise the distance between the end-effector location and the target cutting location.

Deliverable due: Month 30

Contents

1 Bush trimming end-effector V2

As described in D2.3 the version 1 bush cutter was designed, built and tested in the Trimbot2020 garden at several occasions and the results from those experiments led to new ideas and the need for a revised version 2 of the bush cutter. For comparison the following table (Table 1) is constructed to mark the differences between the two versions

Table 1: Comparison V1 and V2 Bush trimming end-effector

	V1	V2
Motor control:	Maxon motor servo control, monitoring of speed and torque. Cables needed for power, Hall sensors and encoders	Speed controller, no monitoring of parameters needed, less cabling
Control:	manual, using the Maxon motion library. Input of start and stop by setting speed values	ROS, switches motor on and off
Motor:	Maxon motor with encoder	Maxon motor, no encoders
Cameras:	mounted on a threaded rod and aluminium structural profile	Mounted in a 3d printed housing
Camera FPGA:	housed in 3D printed box separated from cameras.	Mounted in a 3d printed housing integrated with cameras
Tool:	Motor mounted underneath the tool	Motor mounted on the robot arm, torque is transferred by means of a flexible shaft

Motor control: during the design of the first version it was not clear how fast and with how much torque the motor should operate to cut efficiently through 10 to 12 buxus branches at the same time. During testing it became clear that the V1 was capable to cut through those branches and the correct motor settings were found and could be fixed. In that way, feedback by means of an encoder is not needed in V2. Next to that, this saves on the amount of cabling and therefore weight (about 0.4 kilogram) which was a problem because of the limited payload of the Kinova manipulator. The weight of the V2 tool itself will be around 0.8 kilogram, the V1 tool weighs 2.2 kilogram.

Control: in V1 the motor was controlled using a Maxon motor Windows application in which the motor could be controlled by setting wanted speed setting values and torque limits. In V2, an application in ROS is currently in development that runs in the state machine of the overall robot system. ROS commands will start the sequence of starting and stopping the blades.

Tool: the motor in V1 was mounted underneath the cutting blades. This solution was chosen because of its simplicity in mechanical design but during lab testing it became obvious that the motor was in the way while cutting the lower parts of the bush. Furthermore, the weight of the motor and especially the cabling needed for servo control proved to be too much. As a result one of the DC motors of the Kinova arm was broken. To alleviate this problem the motor in V2 is mounted on the 4th axis of the manipulator. See Figure 2.

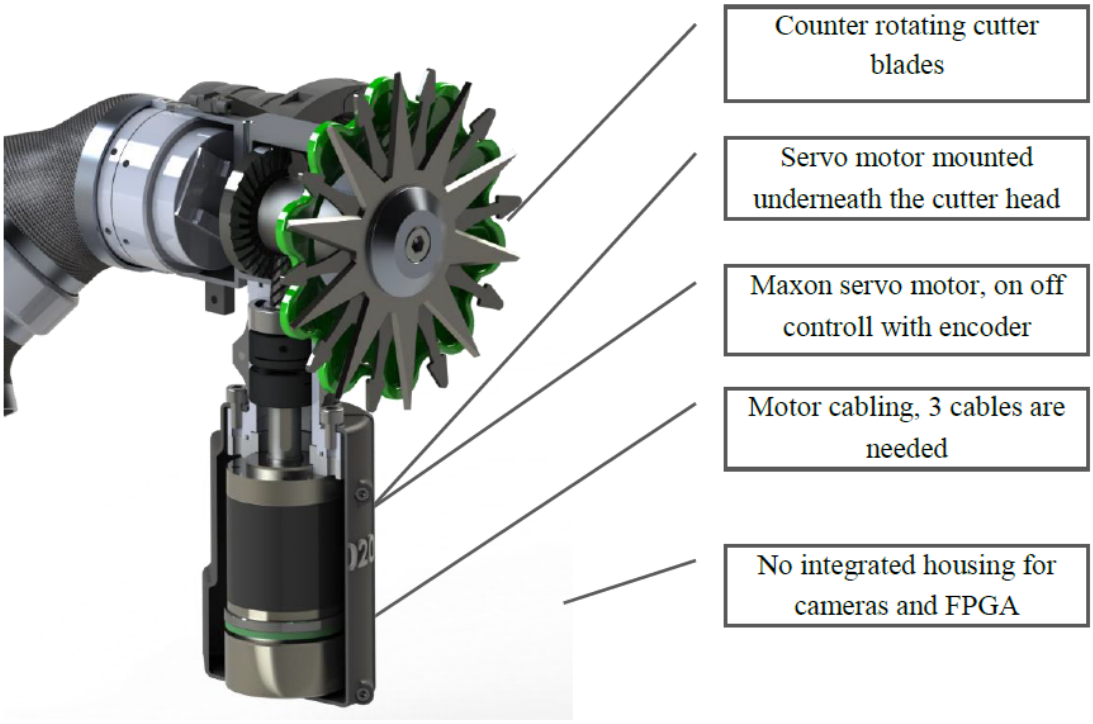


Figure 1: Bush trimmer V1

This distributes the weight over the arm, makes the cutter smaller and less cables are needed to control the motor. The downside is that a flexible shaft is needed between the motor and the cutter head which still needs to be designed.

Cameras: the cameras were mounted on a structural aluminium profile which was susceptible for misalignments and proved to be fragile. Next to that the FPGA was mounted in a separated housing and cables were running from the camera to the housing being unprotected. For the V2 end-effector a new housing for the cameras and FPGA is designed and 3D printed. The housing holds three pairs of stereo cameras as can be seen in Figure 3. This housing is jointly designed by WR and ALUF.

To be able to improve the arm control by means of visual tracking (see section 4.4 for details) one or several April markers will be placed on the end-effector. A possible April marker position on the bush trimmer V2 is shown in Figure 4

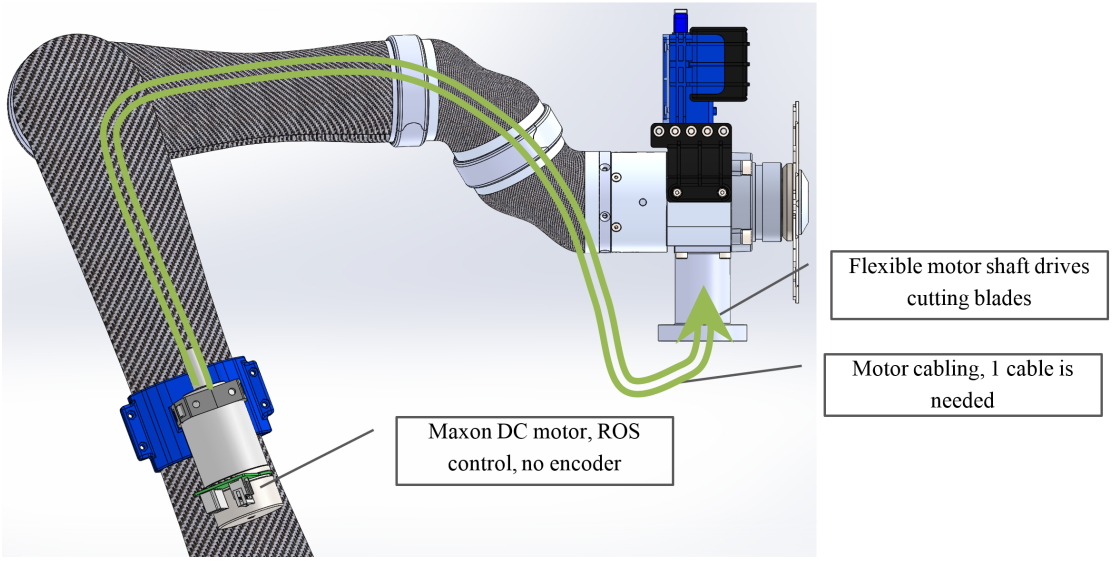


Figure 2: Bush trimmer V2

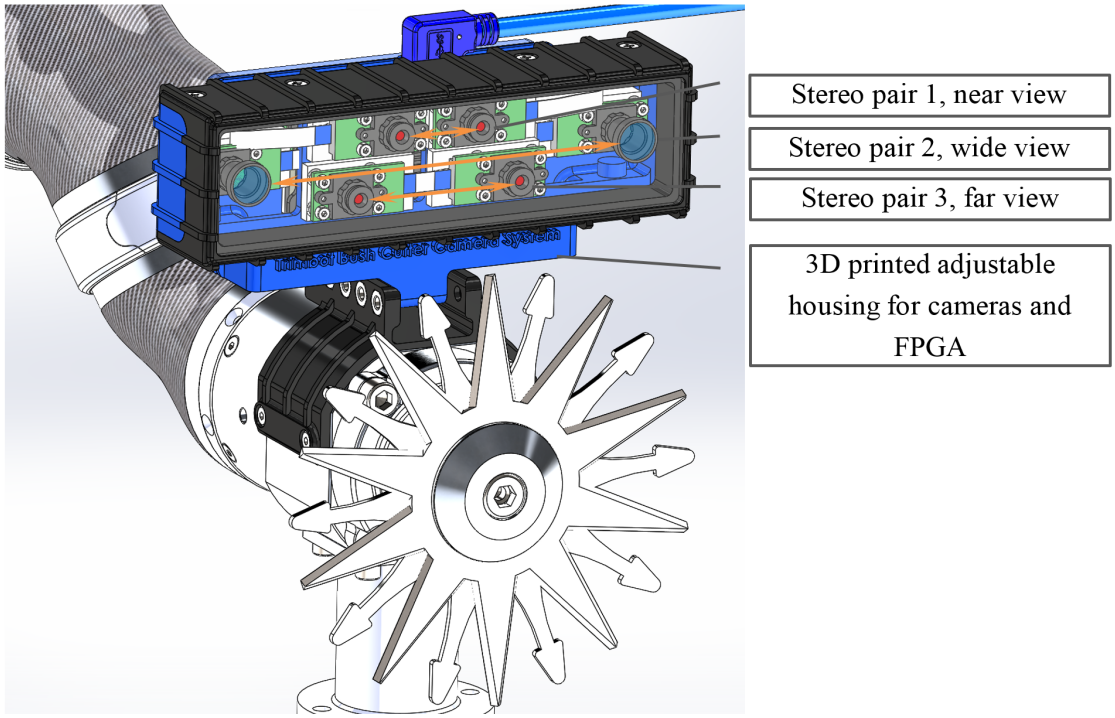


Figure 3: Bush trimmer V2 design camera placement

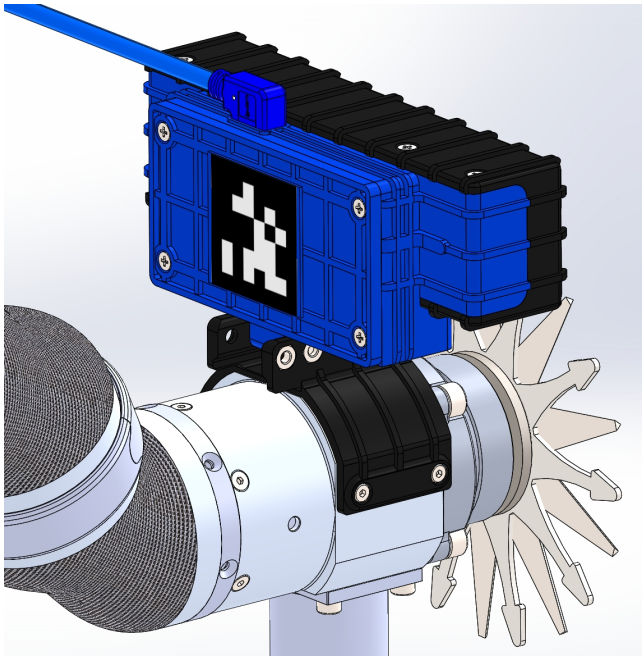


Figure 4: Back of bush trimmer V2 with april tag calibration object.

2 Rose cutting end-effector V2

As described in D 2.3 the version 1 Rose cutter was designed, built and tested in the Trimbot2020 garden at several occasions and the results from those experiments led to new ideas and the need for a version 2 of the rose cutter. For comparison the following table is constructed to mark the differences between the two versions:

Table 2: Comparison V1 and V2 rose cutting end-effector

	V1	V2
Motor control:	Maxon motor servo control, monitoring of speed and torque. Cables needed for power, Hall sensors and encoders	Speed controller, no monitoring of parameters needed, less cabling
Control:	manual, using the Maxon motion library. Input of start and stop by setting encoder values	ROS, monitors on off switches mounted in the cutter head
Motor:	Maxon motor with encoder	Maxon motor, no encoders
Cameras:	mounted on a threaded rod and aluminium structural profile	Mounted in a 3d printed housing
Camera FPGA:	housed in 3D printed box separated from cameras.	Mounted in a 3d printed housing integrated with cameras
Tool:	Cutter head mounted with some flexibility on motor housing	Shorter motor housing, camera housing mounted on tool

Motor control: during the design of the first version it was not clear how fast and with how much torque the motor should operate to cut efficiently through a rose branch of maximum 12 mm. During testing it became clear that the V1 was capable to cut through those branches and the correct motor settings were found and could be fixed. In that way, feedback by means of an encoder is not needed in V2. Next to that, this saves on the amount of cabling and therefore weight (about 0.4 kilogram) which was a problem because of the limited payload of the Kinova manipulator.

Control: in V1 the motor was controlled using a Maxon motor Windows application in which the motor could be controlled by setting wanted encoder values, speed and torque limits. In V2, an application in ROS is currently in development that runs in the state machine of the overall robot system. Two limit switches in the cutter head will monitor whether the knife is fully opened or closed. ROS commands will start the sequence of closing and opening the knife while monitoring the state of the two limit switches.

Cameras: the cameras were mounted on a structural aluminium profile which was susceptible for misalignments and proved to be fragile. Next to that the FPGA was mounted in a separated housing and cables were running from the camera to the housing being unprotected. In V2, a new camera housing with integrated FPGA is designed where two stereo camera pairs can be mounted with different base lengths. The housing is made weather proof by means of rubber seals. The position of the cameras is chosen at the back of the tool where it is connected to

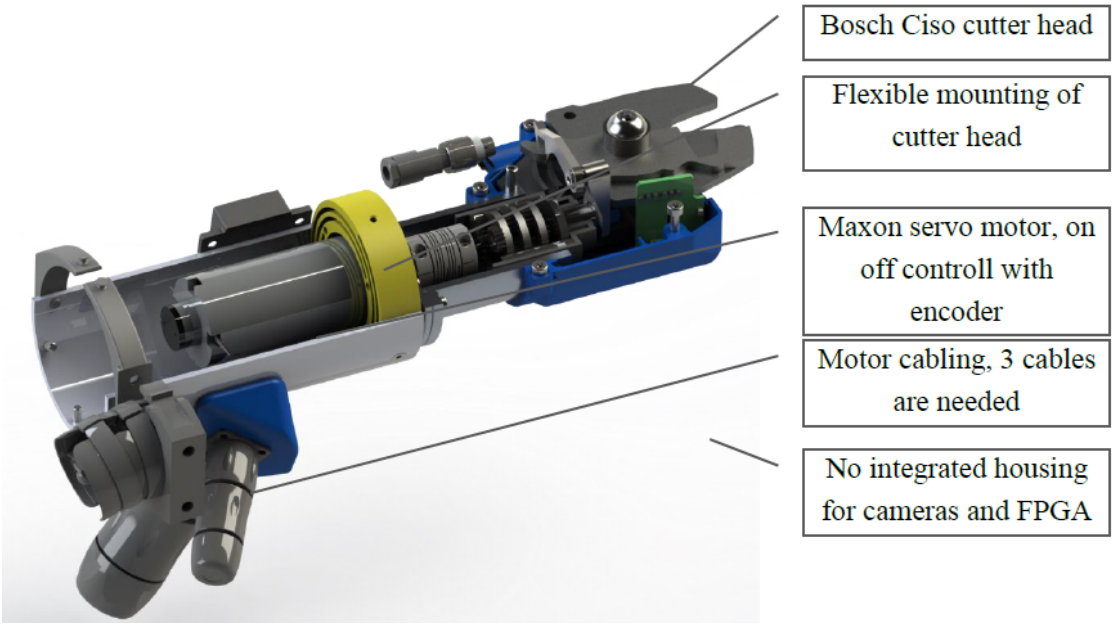


Figure 5: Rose cutter V1

the robot arm. The right height and angle were found using SolidWorks by simulating the FOV of the cameras and finding the right location to have as much of the rose bush in view while maintaining a good sight at the cutter head. See pictures in Figure 9.

To be able to improve the arm control by means of visual tracking (see section 4.4 for details) one or several April markers will be placed on the end-effector. A possible April marker position on rose cutter V2 is shown in Figure 8.

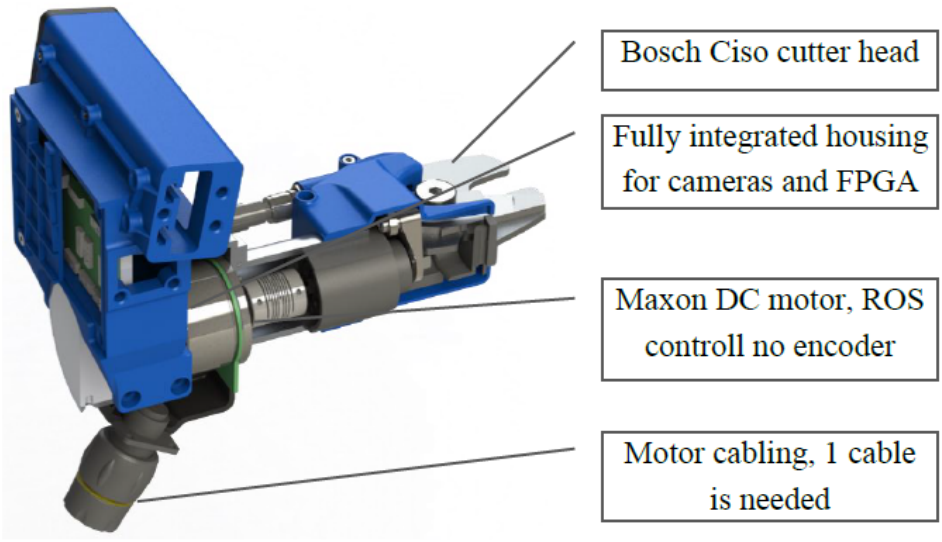


Figure 6: Rose cutter V2 rear view

Figure 7: Rose cutter V2 front view

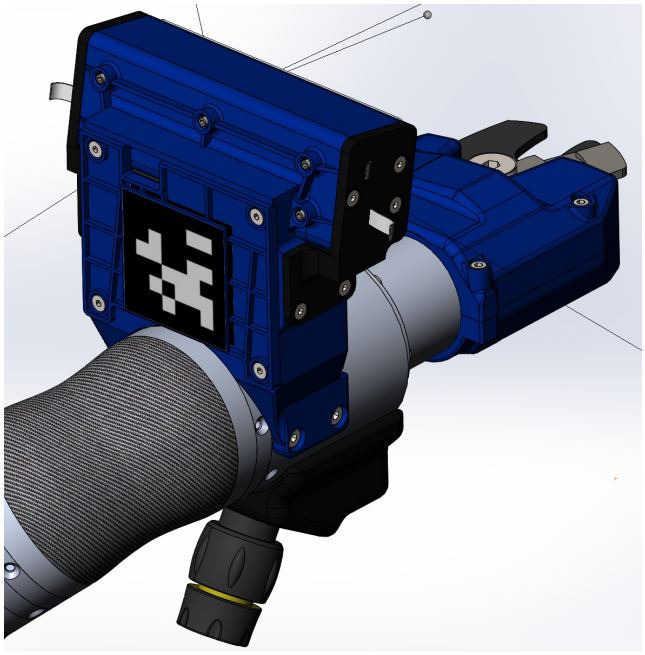


Figure 8: Back of rose cutter V2 with april tag calibration object.

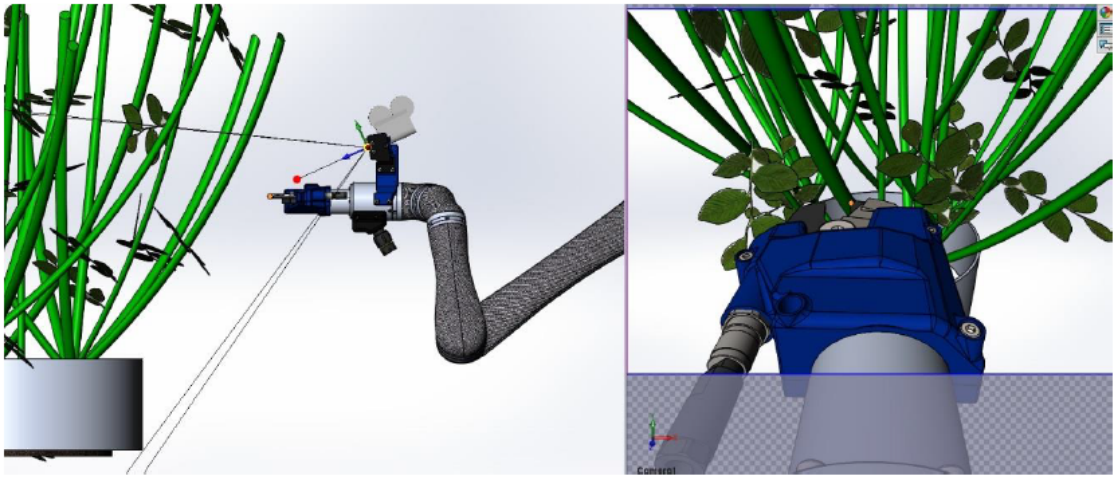


Figure 9: Left side view of the simulated camera on top of the V2 rose cutter. Right; Simulated camera view of the cutter head and the rose bush.

3 Accuracy issues in controlling the Kinova Arm

To have accurate bush trimming, even when using visual servoing to correct for sensing errors, it is required that the robot arm used has sufficient accuracy in its control, so that the desired poses are also actually reached. From previous experiments and testing, including those for deliverable 2.3, some issues with insufficient control accuracy where found. Especially during calibration of the camera setup, it was found that there were large errors in the resulting calibration settings. It was expected that part of these errors originated from the behaviour/properties of the Kinova Jaco robot arm, especially since the used tools were about to exceed the specifications of the arm. In this section, we will explain in more detail what has been investigated in this respect, which results where found, and how this will be dealt with the continuation of the research.

3.1 Accuracy evaluation

To evaluate the accuracy of the arm control and encoder readout, several checks and evaluation actions were executed. These were:

- Checking the encoder response when applying external force or displacement on the arm.
- Test if pose commanded to the arm was actually reached.
- Test the effect of load on reaching the commanded pose.
- Test mechanical properties of the arm.

For the first test, the arm was set to a stretched pose, and an external force was applied to the arm, by manually pushing / lifting the last links. At the same time, encoder readout (showing the rotation of each joint in radians) was visualised to see if the encoder did measure these displacements. It was observed that the encoders responded to even small pertubations of the arm, with orientation changes smaller than 10e-6 being indicated properly. Furthermore, it was found that the construction supporting the arm was rather flexible. To reduce this, first a mechanical fixing by wooden blocks was used, which was replaced later on by an improved mounting construction. As result of this step, it was concluded that given proper fixing of the arm, the encoder data seem to be reliable for showing differences in arm poses. Not tested, however, was whether these values did correspond accurately to the actual orientation of the joints.

In the second step, it was investigated if the arm actually reached the poses that were commanded. For that, a pose command was issued to the arm, and the encoder readouts of the resulting pose were logged. This was repeated over a wide range of arm configurations, also the influence of arm configuration was included. From the results of this action, the difference of expected and actual arm pose could be calculated and visualised as function of joint angles. For controlling the arm, the built-in velocity controller was used, as using fixed-pose control

is not suitable when desiring a moving arm that performs bush trimming. It was found that especially for joint 2 and 3, clear deviations between commanded and actual pose occurred (see also Figure 10. Without load, these differences could be as large as 0.01 radians, leading to a difference in end effector pose of 10 mm (calculated using the kinematic model of the arm). When including loads, maximum deviations increase to 0.017 for joint 2 and 0.007 for joint 3, depending on arm pose. In another trial, the different inverse kinematic solutions were used to reach the same position of the TCP in Cartesian space. From this, opposite offsets in joint angles were observed. Resulting TCP positions had a variation of max 5 mm in the XY-plane, and 6-12 mm in the Z-plane, all being lower than the commanded TCP positions. Such deviations can be explained by gravity, as the arm pose happened to almost always be lower than commanded, and the largest impact of this could be observed on joints 2 and 3 which are mounting points of the first and largest links that move in 3D space. Thus, it was concluded that the arm has difficulty in properly correcting from the gravitation force.

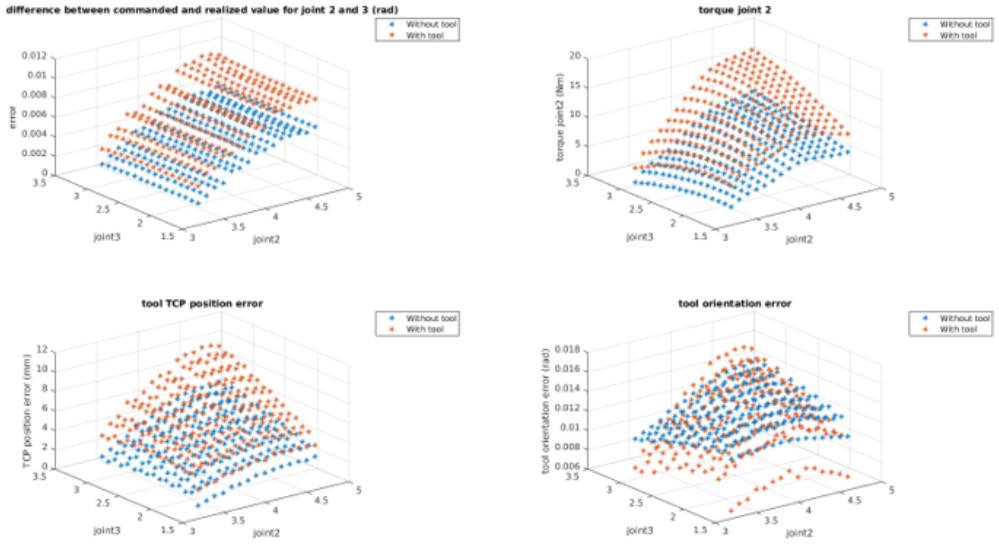


Figure 10: Differences between commanded and actual arm poses

.

Next, in a similar fashion also the effect of adding load (the end-effector) to the arm was investigated. For this, two configurations were tested: - link 2 pointing upwards and link 3 pointing forward - link 2 and 3 both pointing forward. This time, also absolute vertical displacement was measured by attaching a laser pointer to the arm, which projected a cross on the wall. By measuring the (downward) displacement of the cross after attaching a load as well as after manually displacing (lifting) the arm, the offsets could be determined. Also encoder readouts in radians were registered. Measurements here were not highly accurate, but provided values within a few mm's accuracy. In this way, also possible mechanical bending of the arm could be detected, as this would lead to the joint rotations to be different from the measured displacement. Results of this are given in table 3. Checking the observed displacements with the measured joint rotations (including the kinematic model of the robot), showed that these displacements were accurately reflected in the encoder readings and joint rotations. From this, it can be expected that the readings of the arm encoders properly reflect the arm pose.

Table 3: Measured offsets when applying load

	original, without tool	with tool	after tool removal
Configuration 1 (joint 2 rotation in radians):	1.622	1.618	1.621
Configuration 1 (displacement in mm):	0	7	3
Configuration 2 (joint 2 rotation in radians):	4.718	4.724	4.720
Configuration 2 (joint 3 rotation in radians):	3.137	3.134	3.136
Configuration 2 (displacement in mm):	0	12.5	3

Finally, it was checked if any fixed / static mechanical bending of the arm could be observed, possibly as result of deformation by previous overload of the arm. Visual inspection did not show any visual deviations (larger than a few mm between joint ends) in the major links. Also reflectance of linear objects in the arm body did not indicate deformations of the arm.

3.2 Conclusions of the tests so far

- Fixing the arm support improved reliability of control;
- No mechanical bending / deformation is observed when applying a load to the arm;
- Encoders seem to properly reflect changes in arm pose;
- There is a difference between the commanded and actual poses of the arm;
- Arm control has limited capability to correct for gravitational influences on arm pose, such that commanded pose is reached only within limits;
- Observed errors seem to fit within specifications of Kinova Jaco arm:
  - 15 mm absolute precision in arm positioning;
  - 4-5 mm repeatability of arm positioning;
- Not evaluated is the absolute 3D difference between commanded and actual pose, for example as result of inconsistencies in encoder mounting / calibration. As result, for example, it is unclear if a reported horizontal arm pose is actually horizontal, or if there is a small offset. Although this has no direct effect on arm control accuracy, it might be relevant when relating arm / TCP position to other objects in the world.

3.3 Proposed improvements

For dealing with the inaccuracies of arm control, various approaches are proposed, which will be pursued in parallel:

- Adding additional pose estimator on arm (DeepTAM from ALUF) to externally estimate arm pose.
- Add an offset to the control commands, such that the gravitational errors are largely corrected.
- Investigate the options for an improved control method. e.g. by including I-term in controller so gravity will be compensated better.

For the first option, external estimation of the arm pose, the DeepTAM algorithm of ALUF will be used. Two additional wide-view cameras are included to the bush trimmer camera module for this purpose, so the behaviour of the arm in 3D can be properly tracked. As this module also needs proper initialisation, the arm will start from a low pose, such that the base cameras can be used to provide an initial position using April markers on the camera module. In Figure 4 and Figure 8 first design ideas are given on where to place such a marker. Alternatively, inputs from the arm encoders might be used.

For the second option, the difference between command and actual pose will be estimated for a large range of poses. By fitting a model on this data, a correction for gravitational influences can be added to the commands for the arm, such that the actual pose corresponds better to the desired arm pose from the path planning module. For this purpose a calibration with the tool mounted on the arm needs to be carried out. As the two different tools (bush trimmer and rose cutter) are different in weight, tool specific calibrations need to take place.

For the third option, it will be investigated if the arm velocity controller can be improved by adding an I-term that compensates for gravitational influences. This approach concerns a are more fundamental solution to the problem, but might require more effort to have it functioning properly.

Other ideas that might require attention:

- For more static testing or applications (possibly including rose cutting), using more forced position control might be a suitable way to go. For bush trimming, which requires a continuously and smooth moving arm, this is not applicable
- Investigation of camera calibration accuracy given the new knowledge on robot arm accuracies, to see if this is already improved, or if other issues are still present here. This should include the accuracy of the camera calibration itself and the influence of joint angle offsets.

-

4 Vision module for bush trimming

This section describes the vision module that takes raw visual perception data and and generates a target object for the bush trimming planning module. Specifically, this module provides information about *where* the object-to-be-cut is located; the planning module then takes care of *how* this is done.

4.1 Module task

The bush trimming vision module operates after the robot platform has navigated to a trimming target and stabilized its own position. This module's tasks are to (a) perceive the 3D environment around the target bush, (b) identify the target and (c) produce a target shape which is sent to the arm motion planning module for further processing. The target shape is parameterized as a fine triangle mesh.

4.2 Depth sensing

Depth sensing is done using a tool-mounted stereo camera and a neural network which estimates a disparity map for two rectified stereo images (see Fig. 11). For the neural network, we use an advanced version [3] of the DispNet approach [5], trained on synthetic data. Assuming a calibrated camera setup, the disparity map is easily converted into a depth map. For more details, we refer the reader to the deliverable documents D5.1 and D2.2.

4.3 Target fitting

The module has access to information about the type of the target bush (i.e. whether it is spherical, cubical etc.) and its approximate size. This information is used to generate an ideal target shape in the form of an unstructured point cloud.

Once the robot has arrived at a trimming target bush, the arm performs a scanning motion during which camera and depth data are collected and fused into a pointcloud representation of the observed scene. The ideal target shape is then fit into the scene data using an Iterative Closest Point (ICP) method. The scanning process is pre-planned to cover a range of viewpoints, and is not controlled by the vision module.

The deliverable document D2.2 describes an earlier version of the fitting method in which only a single camera pose's depth data was used to fit a target shape. This tends to fail for bushes that are significantly overgrown or whose general shape does not conform well to the desired target shape.

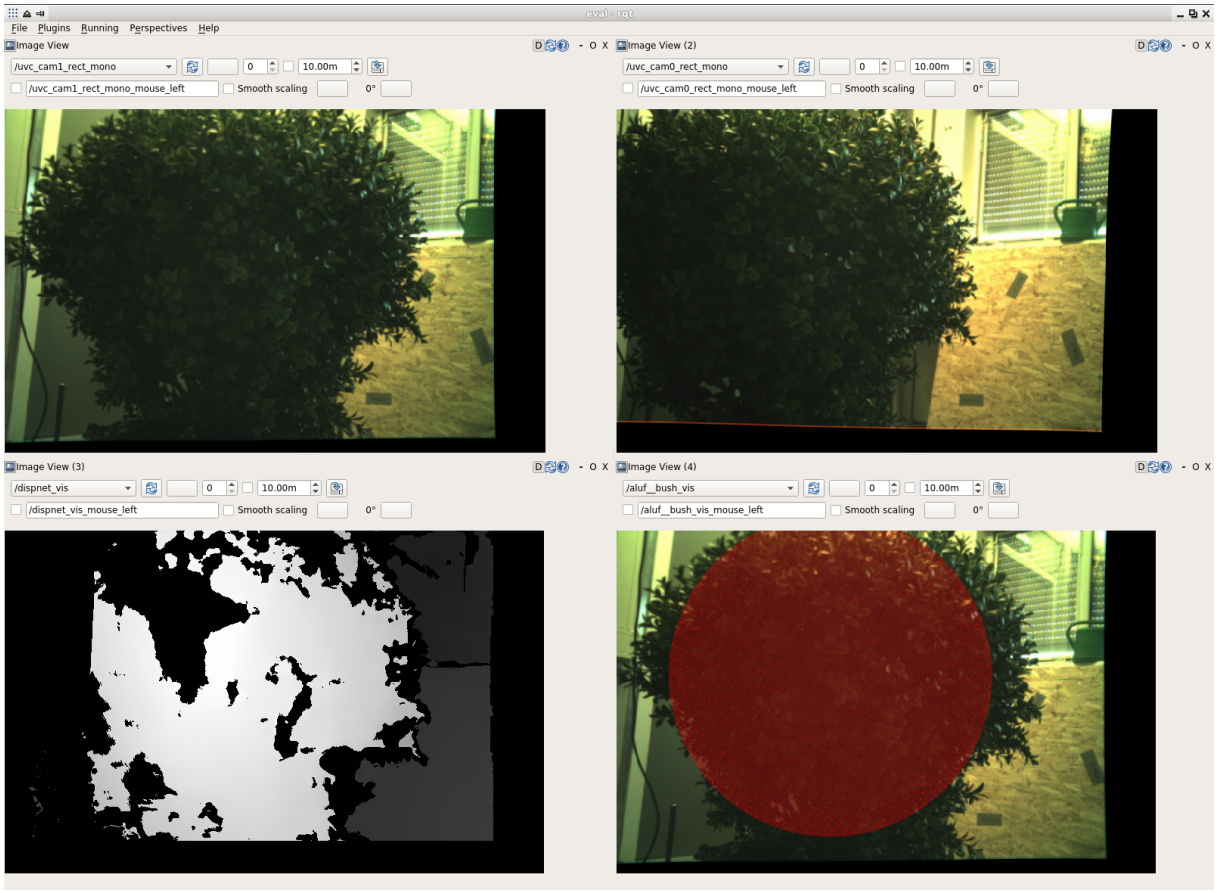


Figure 11: *left to right, top to bottom:* input stereo pair (rectified); consistency-filtered disparity map; reprojected model fit.

4.4 Visual tracking

The robot arm used in the project is lightweight and relatively affordable, but it is also less rigid and accurate than an all-metal industrial model. This holds especially in the low-level arm control, which has difficulty to account for gravitational influences. Especially when a heavy tool load is attached, this leads to the actual arm /tool pose to deviate from the commanded pose (see section 3). Although part of these errors can be corrected, for example by adding an offset to the desired pose that accounts for the gravitational displacement, this still can lead to errors in perception, planning and execution. Especially for trimming evaluation, it is critical that the pre-trimming viewpoint can be reproduced after trimming.

Pointcloud alignment can help reduce the error (see Fig. 13), but it is costly and does not account for the error modes in the depth estimates. Directly tracking using the camera data promises to be more robust.

We are adapting the visual tracking component from DeepTAM [9] to work on depth maps from the disparity estimation component (see Sec. 4.2). This module uses the arm controller's pose data as initialization. Using video data from the arm cameras, it then refines the cameras' positions (and by extension that of the end effector).

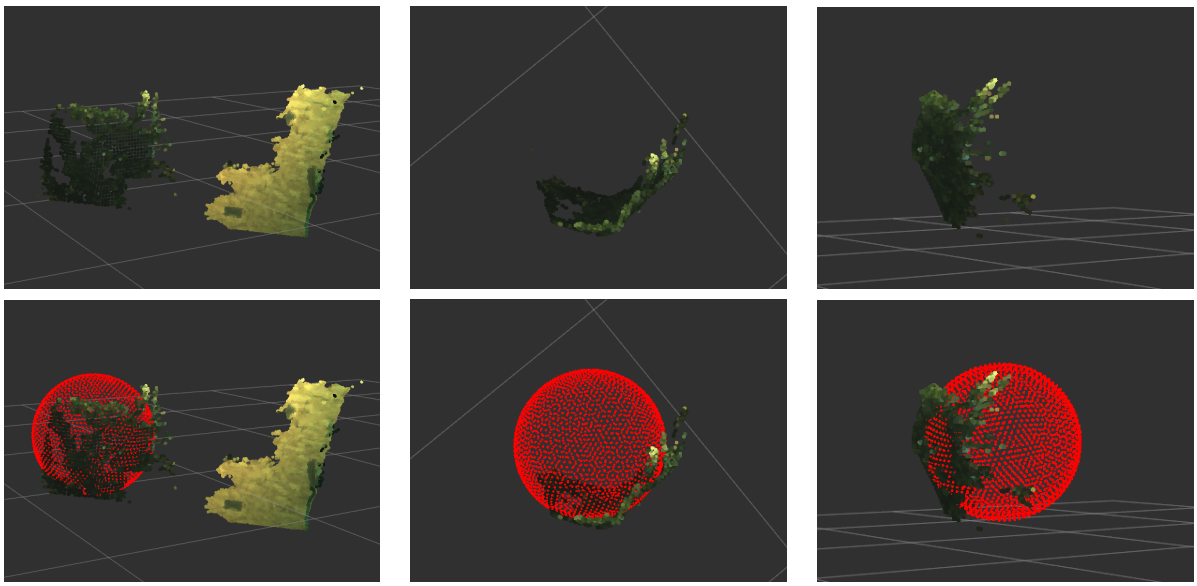


Figure 12: *top row:* multiple views of partial 3D reconstructions from (single view) disparity map (see Fig. 11). *Bottom row:* Same as top row, with added model fit visualization.

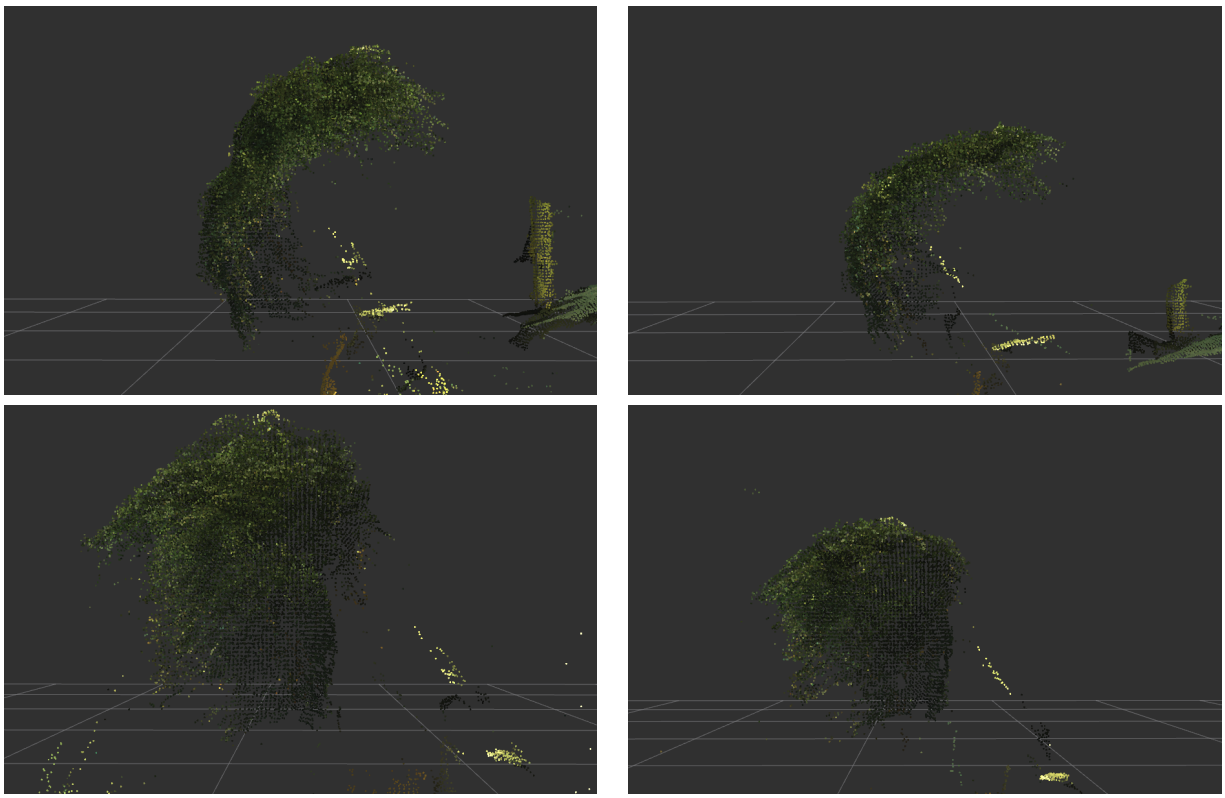
5 Vision module for rose cutting

This section describes the perception process required for the rose pruning task. The desired actions of the robot to prune a rose are cutting thin branches and branches just above the bud eyes. Thus, the existence of a branch, its diameter, and buds have to be detected. The detection is performed in two main steps: branch segmentation and bud eye detection, which will be developed further in the following sections.

5.1 Branch segmentation

First of all, we have to locate the branches of the desired rose bush and obtain their morphology. In Figure 14 we introduce a general workflow for this process.

The workflow consists of segmenting the existing branches of two images captured by a stereo camera using two CNNs. Then, we use the segmented regions as a masks to cover the parts of the images that do not contain a branch. The masked images are used to obtain a normalized depth map of the left camera. The depth map, the color image, and the mask are used to obtain the skeleton of the bush, the branch divisions, and branch width. A detailed explanation for the segmentation can be found in Section 5.1.1. Section 5.1.2 will describe the network that will be used to find the depth map. Section 5.1.3 will explain the methods used in the post-processing block. Finally, we will describe the synthetic dataset that will be used to train the CNNs in Section 5.1.4.



naive pointcloud fusion ICP-aligned fusion

Figure 13: side (top row) and front (bottom row) views of of pointclouds collected over ca. 90 seconds of camera motion. Using only the arm controller's pose data (left column) leads to significant misalignment between the individual views which makes extracing a target shape infeasible. ICP alignment before each frame's fusion preserves the shape better, but this is relatively slow, and hindered by imperfect depth information. Alignment trom tracking could be faster and more robust.

5.1.1 Segmentation

We use an encoder-decoder like CNN architecture [1] with residual connections to segment the branches in an image. This CNN is used to segment the images captured by the stereo camera, one CNN per camera. The output is a binary image which indicates whether or not pixels belongs to the branch or to the background. The CNN is trained using the synthetic dataset described in Section 5.1.4 and fine-tuned with a real dataset. Figure 15 shows an example of the training images.

5.1.2 Depth map

The output of the CNN is combined with the images captured by the stereo camera as a mask, as explained in previous paragraphs. Then, the masked images are used as input to an encoder-decoder like CNN. The output of the network is a normalized depth map of the image captured

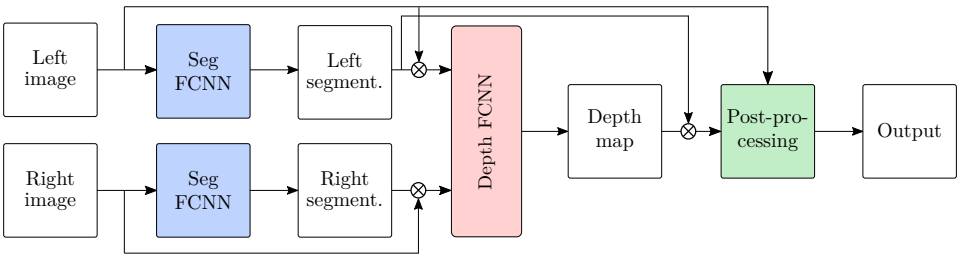


Figure 14: Branch segmentation workflow.

by the left camera. The network is trained following a similar procedure to the previous section. An example of depth images used for the training can be found in Figure 15b.

5.1.3 Post-processing

The segmented depth maps are further post-processed to output a skeleton of the bush (Figure 15d), the diameter of the branches and the nodes of the bush in 2D using the mask and depth obtained by the CNNs. Skeletonization of the bush is a morphological operation that transforms segmented branches into their medial axes, which are assumed to represent the branching topology. Because we want to capture the branching topology better, we do not skeletonize the segmented mask that was found by the CNN. In that case some partially overlapped branches might be reduced to a single skeleton, losing in this way the existence of some branches. Instead, we create a segmented image more suitable for the skeletonization by adding the edges of the plant, obtained by the magnitude of the gradients of the depth map, to our segmented mask.

To find the nodes of the bush, we use the skeleton of the bush and a depth-first search algorithm. It determines if a pixel in the skeleton is a node or a branch by observing its neighborhood. If it has more than two neighbors, the pixel is labeled as a node. It also makes a directed graph between the nodes and the edges that are connected to it, so we can track a branch to its origin at the bottom of the image.

The diameter of the branches is found using the Hough lines transform and the mask created using the depth map. The process is the following: First, we use the Hough lines transform to find lines using the skeleton of the plant. Then, we grow lines perpendicular to the lines found by Hough until they reach the borders of the mask. Because the mask delimits the region of each branch, each perpendicular line ends up being the diameter of a branch in 2D. We can easily find the diameter in 3D by using the stereo camera intrinsics.

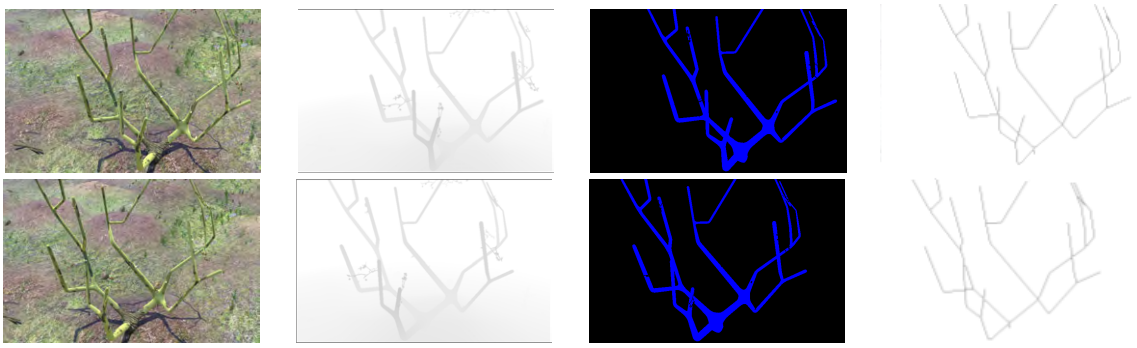
5.1.4 Dataset

Using the 3D creation software blender, we generated 2880 image pairs of synthetic rose bushes with the following characteristics:

- 36 different rose bushes (Figure 16),

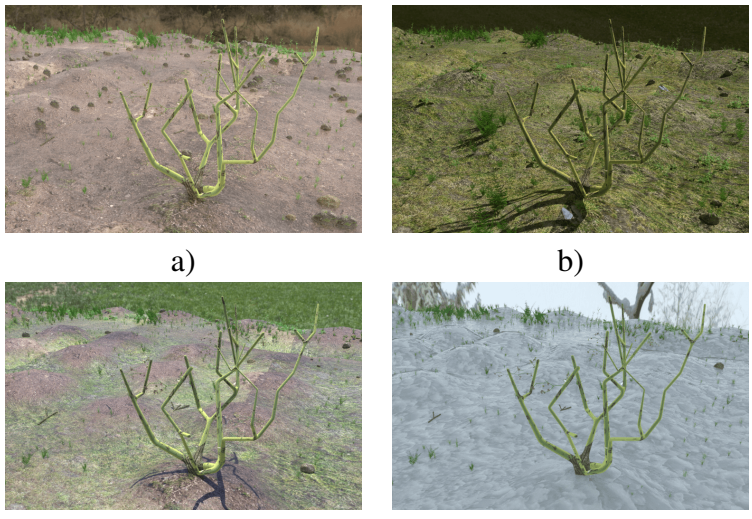
- 4 different environments, 9 bushes per environment,
- 80 image pairs (Figure 15a),
- depth map (Figure 15b),
- pixel-wise segmentation (Figure 15c),
- plant skeleton (Figure 15d).

An example of the dataset can be found in Figure 15 and an example of the 4 environments in Figure 16.



a) color image b) depth map c) segmentation d) skeleton

Figure 15: Synthetic dataset example. Top row shows the left camera view and the bottom shows the right camera view.



a) b)

c) d)

Figure 16: Different environments in our synthetic dataset.

5.2 Bud eye detection

We detect the bud eyes using a CNN with an architecture similar to YOLO [7]. The output of the CNN is a grid cell where each cell indicates the probability of the existence of a bud-eye and a bounding box, as seen in Figure 17. The detection is performed in 2D, however, we can easily find the location of the object in 3D wrt. to camera by using the stereo camera intrinsics. Thus, the final result of this step is a list with the 3D locations of all bud eyes where cutting might take place.

Figure 17: Bud eye detection.

5.3 Bud eye selection / determination of cutting locations

The result of the previous step is a list of all bud eyes locations that are potential cutting locations. As result, a single branch can feature multiple bud eyes, while on each branch, preferably a single cut is made. Some additional selection is needed to reduce the full list of bud eyes found to a set of bud eyes that actually are determined as cutting locations. Several approached exist for this:

1. Manually indicating the bud eye to determine the location that should be cut
2. Selecting only the lowest bud eye on each branch
3. Including plant and gardening knowledge to determine at which bud eye the branch should be cut.

For the first tests, method 1 will be used, by simply selecting the bud eye locations to be cut / taking out all other bud eyes. The final Trimbot2020 system, however, should implement more advanced methods as it is what is actually desired and also used by gardeners.

6 Trajectory control for bush trimming

In the current implementation of the autonomous trimming system for TrimBot2020, the target bush is scanned once, a coverage trajectory plan is computed based on the acquired 3D data, and then such trajectory is executed a number of times in order to cover the whole plant. A rotation table is exploited to allow the arm to reach every side of the bush (check Deliverable 2.3 for a detailed description). This stationary setup can be naturally extended to a mobile system by computing a set of robot base poses that allow to cover the plant, and allowing the robot to reposition itself with respect to the bush frame.

6.1 Issues with open-loop trimming

Repeating the same trajectory on multiple plant sides would make sense if the originally acquired target shape model were reliable in terms of position and size, and if the motion execution carried out during the first run had displayed a satisfying cutting behaviour. Both conditions are actually not guaranteed: several sources of errors may make the computed trajectory totally inadequate for the task. Repeating such trajectory would symmetrically spread the errors introduced by the initial run. For instance, if the acquired bush model has been fitted with a translation error, the robot may cut too deeply only on one side of the plant. But by rotating the plant and repeating the motion, the plant would be cut too deeply everywhere. Passing from a stationary to a mobile manipulator would introduce further uncertainties.

6.2 Adaptive trimming concept

If an attempted trimming trajectory turns out to be unsuitable for the task, it should not be repeated. This leads to two main questions: 1) how to evaluate the quality of a trimming task on the plant portion involved in trimming, 2) how to produce a new trajectory plan that is likely to result in a better trimming behaviour in case the perceived trimming quality was unsatisfactory. This section focuses on the second question. One possible idea to tackle the first issue is to exploit a silhouette-based bush profile evaluation system, by limiting it only to the angular range interested by the trimming plan (which can be estimated beforehand).

It is reasonable that a better performing trajectory can be estimated from the initially computed one, i.e. the arm trajectory spanned an area that is somehow correlated to the real target boundary. Thus, the trimming planning module should be adaptive, and able to gather information about how well we trimmed. So the error between the trimmed profile and the desired profile has to be mapped to a perturbation in the trajectory plan. The idea is to update trajectories according to some reasonable interpretation of the trimming effect (e.g. inflate the target set of patches if too much deep cut is detected, shift the poses if the trimming effect is shown only on one side).

Thus, we want to exploit the sensed knowledge of how we performed our trimming task to *warp* an existing trajectory plan. We also consider the opportunity to exploit all the *experience* we

acquired about previous trimming trajectories to produce new trajectories for new scenarios. This would be closed-loop among different trimming operations. It may take a lot of exemplar runs to perform proper parameters learning, but the mechanism will be flexible enough to adapt autonomously to modifications in the trimming settings.

In Figure 18, a diagram of the proposed trajectory update system is shown.

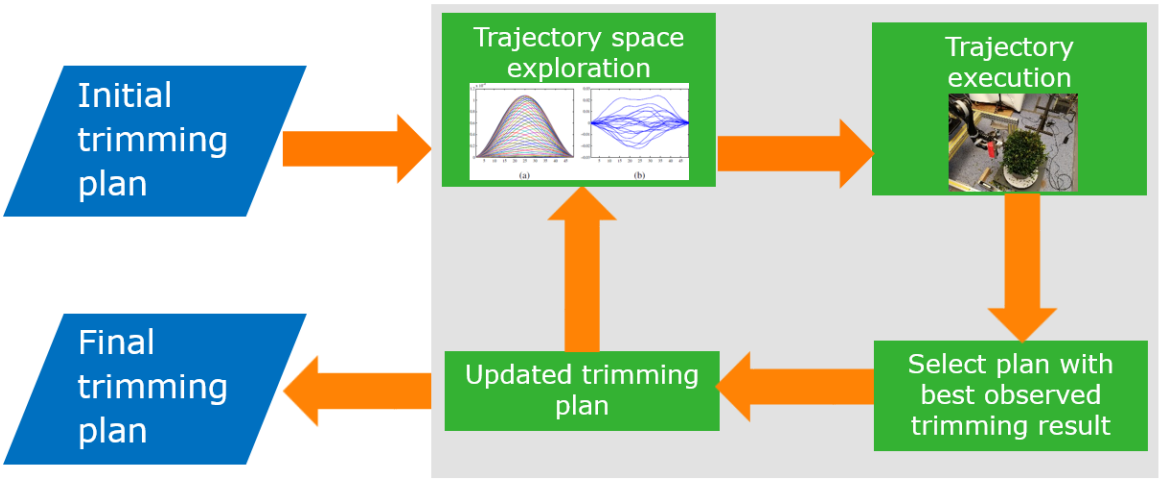


Figure 18: Concept of trajectory adaptation system based on observed trimming score.

.

6.3 Implementation

In this section the system used to perform trimming result-based plan adaptation is described. The algorithm is based on a Reinforcement Learning approach called Policy Improvement with Path Integrals (PI^2) [8].

6.3.1 PI^2 framework

The trimming task is modeled as a dynamical system of the form:

$$\dot{\mathbf{x}}_t = \mathbf{f}(\mathbf{x}_t, t) + \mathbf{G}(\mathbf{x}_t)(\boldsymbol{\theta}_t + \boldsymbol{\epsilon}_t) \tag{1}$$

where $\mathbf{x}$, is the system state (i.e. the arm configuration at each time step), $\mathbf{f}(\mathbf{x}_t, t)$ is the passive dynamics, $\mathbf{G}(\mathbf{x}_t)$ is the control transition matrix, $\boldsymbol{\theta}_t$ is the control input at each time step, and $\boldsymbol{\epsilon}$ is Gaussian noise.

The *policy parameters vector* $\boldsymbol{\theta}$ consists of a time-discretized sequence of arm configurations composing the desired trajectory. The considered dynamical system instantiation is a proportional control to track the discretized trajectory:

$$\dot{\mathbf{x}}_t = \mathbf{K}(\mathbf{g_t}^{\mathrm{T}}(\boldsymbol{\theta} + \boldsymbol{\epsilon}) - \mathbf{x_t}) \tag{2}$$

where $\mathbf{g}_t$ contains 1 at index t and 0 at all other indices.

The state-dependent cost to be optimized has the form:

$$r_t = q_{\mathbf{x}}(\mathbf{x}, t) + \frac{1}{2}\boldsymbol{\theta}_t^{\mathrm{T}}\mathbf{R}\boldsymbol{\theta}_t \tag{3}$$

where $q_{\mathbf{x}}$ is an arbitrary state-based cost function and $\mathbf{R}$ is a matrix to control the generated policy complexity (see section 6.3.2)

The total cost of a trajectory is given by the sum of the rewards at each time step, plus a terminal cost ϕ_{t_N} representing the observed task quality at the end (see section 6.3.3):

$$S(\tau) = \sum_{i=1}^{N-1} r_{t_i} + \phi_{t_N} \tag{4}$$

To optimize the policy parameters $\boldsymbol{\theta}$, PI^2 explores feasible trajectories that improve the cost through iterations. In the following the pseudocode of one iteration of the procedure is reported.

- Create K rollouts of the system from the same start state using stochastic parameters $\theta + \epsilon_t$ at every time step;
- For $k = 1...K$, and time-steps $i = 1...N$, compute the state-based cost $S(\tau_{i,k})$. The perturbation ϵ_t is projected under the metric $\mathbf{R}^{-1}$, thus binding the explored trajectories (see Section 6.3.2 for details on $\mathbf{R}$).

$$\mathbf{M}_{t_i} = \frac{\mathbf{R}^{-1}\mathbf{g}_{t_i}\mathbf{g}_{t_i}^{\mathrm{T}}}{\mathbf{g}_{t_i}^{\mathrm{T}}\mathbf{R}^{-1}\mathbf{g}_{t_i}} \tag{5}$$

$$S(\tau_{i,k}) = \phi_{t_N,k} + \sum_{j=i}^{N-1} q_{t_j,k} + \frac{1}{2}\sum_{j=1}^{N-1}(\boldsymbol{\theta} + \mathbf{M}_{t_j}\boldsymbol{\epsilon}_{t_j})^T\mathbf{R}(\boldsymbol{\theta} + \mathbf{M}_{t_j}\boldsymbol{\epsilon}_{t_j}) \tag{6}$$

- The state-based costs for each trajectory are used to compute the contribution $P(\tau_{i,k})$ that a given trajectory provides to the policy update. Such contribution is defined as exponentially lower for higher costs, so that only low-cost trajectories significantly contribute to the policy update.

$$P(\tau_{i,k}) = \frac{e^{-\frac{1}{\lambda}S(\tau_{i,k})}}{\sum_{j=1}^{K} e^{-\frac{1}{\lambda}S(\tau_{i,j})}} \tag{7}$$

- For time-steps $i = 1...N$, compute

$$\delta\boldsymbol{\theta}_{t_i} = \sum_{k=1}^{K} P(\tau_{i,k})\mathbf{M}_{t_i,k}\epsilon_{t_i,k} \tag{8}$$

6.3.2 Control complexity cost

The main issue when using the proposed policy representation is that adding noise directly to a discretized trajectory is likely to make the perturbed trajectory jerky and difficult to execute on a real robot. For this reason, a control jerkiness-dependent cost is added to the task-based trajectory cost, via the quadratic cost matrix $\mathbf{R}$.

Analogously to other works on motion plan optimization like [4, 2] the control matrix is defined as a weighted sum of difference matrices of order d:

$$\mathbf{R} = \sum_{d=1}^{D} w_d \|\mathbf{A}_d\|^2 \tag{9}$$

To make the control cost $\boldsymbol{\theta}_t^{\mathrm{T}} \mathbf{R} \boldsymbol{\theta}_t$ equal to the sum of squared accelerations along the trajectory, the difference matrix of second order $\mathbf{A_2}$ is used:

$$\mathbf{A_2} = \begin{bmatrix} 1 & 0 & 0 & & 0 & 0 & 0 \\ -2 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & -2 & 1 & & 0 & 0 & 0 \\ & & \ddots & & \vdots & & \ddots \\ 0 & 0 & 0 & & 1 & -2 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -2 \\ 0 & 0 & 0 & & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

6.3.3 Trimming outcome-based rewards

The trimming outcome-based reward functions are:

- Percentage of the target surface covered by trimming (see definition of Trimming Performance in Deliverable 2.3)
- Trimming accuracy (i.e. how well the trimmed profile falls within an acceptability ring around the target shape, see definition in Deliverable 2.3)
- Smoothness of the trimming error (i.e. how regular is the deviation between the trimmed profile and the target profile)

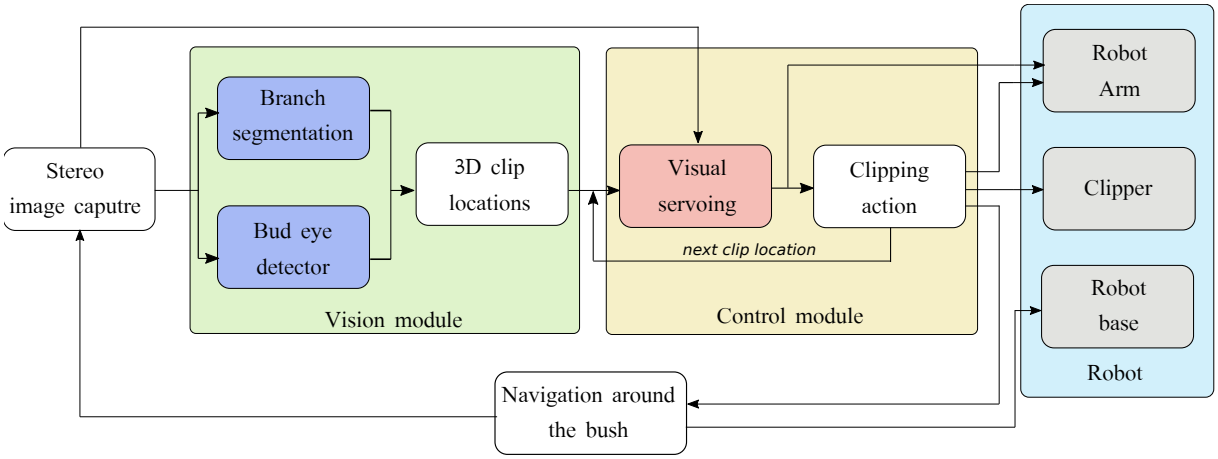


Figure 19: Trajectory control workflow by visual servoing.

7 Trajectory control for rose cutting

This section describes the trajectory control for the rose cutting task. The goal of this task is to make the robot cut rose bushes according to the best practices for seasonal rose pruning, for example branches slightly above a bud eye and weak (thin) branches. Here, we use a calibrated stereo camera mounted on the end-effector of a robotic arm (eye-in-hand); the end-effector is a cutting tool. This task can be modeled as a grasping problem so we can use a visual servoing approach to get close to the branches. In figure 19 we can observe a general workflow of the steps necessary to perform the visual servoing.

The workflow starts with the branch segmentation and bud eye detector steps, which were covered in the visual module for rose cutting (Section 5). The outputs of these two steps are points in 2D that show which part of the branches should be cut. The 2D clip locations together with the intrinsics of the stereo camera are used to find the corresponding locations in the 3D coordinate system of the camera.

All detected locations are queued and sequentially sent to the control module, one by one in a top-down, left to right order, to make it easier for the robot to cut the branches of the bush. We cut the external branches first, and then go deeper. To cut a branch slightly above an eye-bud, we send the location of the eye-bud plus a fixed distance (2 cm) above it along the branch as the target for the servoing mechanism.

For each received clipping site the visual servoing algorithm outputs a visual-motor policy that allows the robot arm to move from its initial position to the desired part of the branch to cut it. Once it reaches the desired pose, we rotate the end-effector 45°and cut the branch by actuating the clipper. Subsequently the clipper is opened to release the branch, the arm returns to its initial position outside of bush, and the process repeats.

To cut the branches that are thin, we use the diameters that the branch detector measures. From the stereo camera we can obtain the real diameter of the plant using the relative 2D diameter found by our algorithm. If it is smaller than a threshold, we send the location of the base of the segmented branch as the point to be cut. Once all branches reachable from the current robot

location are cut, the robot will move around the rose bush in clock-wise direction, keeping a constant distance from it.

In the following section we describe in more detail the visual servoing method to control the robot.

7.1 Visual servoing

The visual servoing control idea for the Trimbot2020 project is to use a reinforcement learning algorithm, more specifically, an Asynchronous Advantage Actor-Critic (A3C) [6] as a visual servoing controller. At the time writing this deliverable this algorithm is still under development.

Reinforcement learning (RL) algorithms learn an optimal policy to solve a task by trial and error and is usually used when a task is difficult to model or has infinite possible states. The main components of an RL are the states of the environment, the reward received after making an action, and the action itself. The RL algorithms can be classified in three categories: policy based, value based, and actor-critic methods, which combine both previous methods and generally perform better than the other two. A3C as an actor-critic algorithm updates its policy with the help of learned state-value functions. However, A3C uses so-called *advantage* function, which measures the advantage of taking action a, over following the policy π at the given time step t, to improve the result. A3C also asynchronously executes different agents in parallel on multiple instances of the environment and updates to a master network after t time steps.

In practice, an A3C is useful for visual control because it uses recurrent neural networks to learn temporal dependencies and visual cues. This guides the robot to make correct movements towards the target and achieve successful grasping (cutting) based on past states. The architecture of an A3C is shown in Figure 20.

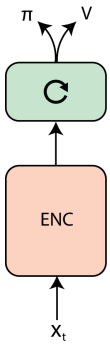


Figure 20: A3C architecture. It receives an image and other useful information of the robot as input or current state x_t. Then, x_t is encoded by a CNN (red block). Finally, the encoded vector is sent to a RNN (green block). The output of the network is a policy π and the value function V.

In our problem, the policy π will be the optimal action (joint movement) the robot has to follow to cut a branch and the value function V will be the expected value of following policy π when

the robot is in the state x_t. Now we will define the three components: state, actions, and rewards of our algorithm.

State x_t

- 6 DoF Joint values,
- 3D Cartesian end-effector location,
- Distances between the end-effector location and locations of all joints (to avoid self-collision),
- Distance between the current end-effector location and the target location,
- RGB-D images.

Action π

- Joint angle movement (6 degrees of freedom).

Reward

- Negative reward:
  - Time from the start of the action,
  - Distance between the end-effector location and the target location,
  - Hit an obstacle,
  - Hit itself.
- Positive reward:
  - Reach the target (terminate).

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[2] Enric Galceran. Coverage path planning for autonomous underwater vehicles. *PhDs Thesis. Universitat de Girona*, 2014.

[3] Eddy Ilg, Tonmoy Saikia, Margret Keuper, and Thomas Brox. Occlusions, motion and depth boundaries with a generic network for disparity, optical flow or scene flow estimation. In *European Conference on Computer Vision (ECCV)*, 2018.

[4] Mrinal Kalakrishnan, Ludovic Righetti, Peter Pastor, and Stefan Schaal. Learning Force Control Policies for Compliant Robotic Manipulation. *International Conference on Machine Learning*, pages 4639–4644, 2012.

[5] Nikolaus Mayer, Eddy Ilg, Philip Häusser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas 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.

[6] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[7] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[8] E Theodorou, J Buchli, and S Schaal. Reinforcement learning of motor skills in high dimensions: A path integral approach. *Robotics and Automation*, 1(3):2397–2403, 2010.

[9] Huizhong Zhou, Benjamin Ummenhofer, and Thomas Brox. DeepTAM: Deep tracking and mapping. In *European Conference on Computer Vision (ECCV)*, 2018.

