Automatic car trunk vacuuming robot

1st Demi Breen
Mechanical Engineering
TU Delft
Delft, Netherlands
d.breen@student.tudelft.nl

2nd Sander Schipper
Mechanical Engineering
TU Delft
Delft, Netherlands
a.p.schipper@student.tudelft.nl

3rd Pepijn Westland
Mechanical Engineering
TU Delft
Delft, Netherlands
p.westland@student.tudelft.nl

4th Suzanne Wiersma
Mechanical Engineering
TU Delft
Delft, Netherlands
s.wiersma-1@student.tudelft.nl

***Acknowledgements*—We would like to thank M. Wisse, M. Klomp, G. van der Hoorn, D. Schepers and D. Kroezen for helping us realise this project.**

***Abstract*—In this research, the Robot Operating System (ROS) was used to create an application for vacuuming the trunk of a car. Knowledge of ROS and the Godel packages was needed to reach the goal of automating the vacuuming of a car trunk. These packages were used to create a simulation and a demonstration, showing the process of path planning, collision avoidance and controlling the robots movements. The results show potential for further development in the field of automation.**

I. Introduction

This research study has been commissioned by Loogman, a Dutch company specialised in the cleaning of cars. Loogman has a fully automatic car wash offering multiple cleaning services. Besides the cleaning of the outside of the car, there is also the possibility to clean the inside. People can use vacuums and cleaning cloths themselves or they can use Loogman's semi-automatic interior cleaning system. The current system contains a conveyor belt and vacuum cleaners hanging from the ceiling. It is similar to a production line; each employee has their own tasks and cleans a specific part of the car. The interior cleaning is high strain manual labour, which results in a high turnover rate of the staff working there. That is why the innovation department of Loogman, called Innovation Lab, reached out to the TU Delft.

When tasked with this problem, the objective was to explore automating the process of cleaning the interior with the use of robotic arms. This research is narrowed down into a first step, starting with only automating the vacuuming of the car trunk, specifically the floor of the trunk. That is why the goal of this project is to perform a demonstration of a robot arm automatically vacuuming the floor of a car trunk, showing the feasibility of the project.

This research study has been conducted as part of the Bachelor End Project for Mechanical Engineering students at the TU Delft in the third year of their Bachelor studies.

II. State of the art

This section covers a quick overview of the current state in the field of robotics and software currently used in the industry.

A. Industrial robots

Automation with robotic arms is widely used, for example in the manufacturing industry. However, there is a big difference between industrial robots and, in this case, cleaning robots. The difference being that industrial robots work in a static environment, while cleaning robots work in a dynamic environment. Because of that, the industrial robots have a pre-programmed path, which stays the same for the lifetime of the robot. In the case of a car trunk vacuuming robot the work piece (the trunk) is different for every car, so the arm cannot repeat the same path every time. It needs to scan the trunk and plan its vacuuming path accordingly to its dimensions.

B. Software

There is an approach designed for flexible robot handling named Scan-N-Plan [14]. Scan-N-Plan contains a set of tools that enable real-time robot trajectory planning from 3D scan data. Software that uses this approach will plan a custom solution based on the scan that has been given as input. The scan and plan parts of this software can be used individually. The scan part contains the scanning of the physical object. The scanner sends the data to the used program, which in turn generates a 3D model. The plan part consists of the path and motion planning of the robot. To use the plan part it is necessary to have a 3D model. Instead of using the scan part, it is also possible to insert a pre-programmed 3D model. This way, the two parts can be used separately. Since there is a limited time for this research study, it will only focus on the planning part of the Scan-N-Plan approach.

The Scan-N-Plan approach is used in applications similar to this project that need custom handling of different products. This software is currently under development. The approach and the existing knowledge in this field will be used as the base for this research, narrowing it down to the specific application of car trunk vacuuming.

When looking into existing software for the Scan-N-Plan approach, two options where found; Bezier and Godel. Bezier is able to generate rectilinear trajectories on 3D surfaces [2]. Another option is the Godel package [7], this package is also useful for trajectory planning, but can execute these with a connected robot as well. These surface trajectory planners are developed for the use of surface blending, which is similar to the case at hand. With Godel being more extensive, the

decision was made to use Godel instead of Bezier. Therefore the Godel package will be used in this application.

III. Theory

This section contains basic information about the software that has been used in this project. This will give the background information for the methods section where the use of these programs is explained.

A. ROS

A big part of this research is understanding and implementing Godel and ROS. The software is quite extensive and complex. Basic knowledge of ROS is required to start using the Godel package. ROS is a widely used framework, which enables people to create robot software. It is made of different libraries, tools and conventions that provide the basics for writing robot software [11].

For this project it is convenient to use ROS with Linux, making the basic use of Linux and its installation functions crucial. In order to be able to use the Godel package, the correct distribution of ROS has to be installed. The Godel package is currently supported by multiple distributions of ROS. ROS Kinetic was chosen as distribution in this project because it is a long-term supported release and at the start of this project the Kinetic-devel branch of Godel was also more stable and reliable than on other distributions. The long-term support makes it usable for future developments of this project. ROS Kinetic is best supported on Ubuntu 16.04, therefore making this the version of Linux that has been installed [12].

ROS uses mostly Python and C++ code to control robots in different ways. It is a large collection of different computer processes, called nodes in ROS. Each node is a small script that has a couple of functions and together these nodes can create advanced programs to control robots.

B. Godel

As mentioned before, the Godel package is used for this project. This is a collection of nodes designed to control robotic arms for the application of surface treatment.

Godel can accept point clouds. Point clouds are the collection of points that define the location of solid objects within a given space. The pre-defined method Godel uses to get this data is via an Ensenso scanner. An Ensenso scanner is a stereo 3D camera that scanss an object and allows to generate a 3D point cloud [5]. As explained before, the use of scanners will be ignored in this project. Instead the point cloud data will be supplied by pre-defined models.

A pre-defined car trunk model has been made by measuring the dimensions of the car by hand and then creating a digital model in SolidWorks. Converting this model to point cloud data can be done with a program called Cloud Compare [6]. This will generate a pre-defined amount of points of a mesh model.

The Godel program then uses this data to generate the tool path for surface blending. Vacuuming is a similar process, because both methods are surface treating and both require surface trajectory planning. Following up on these similarities, Godel will be made suitable for the use of vacuuming the trunk of the car.

C. MoveIt

MoveIt is a program that can be used on top of ROS. It is an open source software package that allows for the control of robots via ROS. It can be used for creating the kinematics of the robot, collision checking and path planning. MoveIt is an important part of ROS when combined with the Godel package. The graphical user interface (GUI) of MoveIt is the MoveIt Setup Assistant, this can be used to create robot configurations [10].

D. OpenRAVE

OpenRAVE has been developed to create and simulate an analysis of kinematic and geometric information related to motion planning of robotic arms. This means that it can generate forward and inverse kinematics solvers, which are used by the Godel software [4].

E. Descartes

Descartes is a package used for path planning. It is used to convert Cartesian paths to joint path plans. It will also work with under-defined paths, which is the case with planning the vacuuming of the car trunk. The start position of the robot and the start position of the path are often not connected and need to be planned, Descartes takes care of planning these movements [9].

F. Octomap

Octomap is a data structure used for collision avoidance. It converts, in this case, a point cloud into voxels in a 3D grid map approach. These voxels are used for the collision detection by the Descartes path planner, to update its trajectory path to check if a collision occurs. The collision avoidance of Octomap adds functionality to the default collision avoidance supplied by MoveIt. This makes it useful for moving the arm inside an enclosed area with many collision surfaces, such as a trunk [16].

IV. Methods

Godel poses a set of challenges in order to be suited to vacuum a car trunk. These challenges will be further explored in this chapter, but will be briefly summarised. The current robot arm used by Godel is not suitable, the tools on the arm have to be changed and the path has to be changed as well. Another challenge is the use of an enclosed work space. The robot arm needs to work on the inside of the car, avoiding collision with the walls of the trunk. The vacuum tool tip may not damage the car while vacuuming the trunk. To solve these challenges, certain packages and files need to be changed or even replaced. For more detailed information regarding the changes, please consult the appendix.

A. Installing Godel

When starting with this project a stable version of Godel was needed. This version can be found on GitHub and is provided with a clear manual for downloading. When building the packages there will be certain errors caused by faults in two Descartes files. In the files:

- utils.h, located at: src/descartes/descartes_core/include
- ikfast_moveit_state_adapter.cpp, located at: src/descartes/descartes_moveit/src

The log_error and log_inform commands are used. Those need to be replaced with ROS_ERROR and ROS_INFO respectively.

B. Implementing a new robot arm

After installing Godel, a new arm has to be simulated, since the standard ABB IRB2400 is not sufficient for this application. The IRB2400 is not sufficient because it is relatively small compared to a car trunk, and the IRB2400 does not have a hollow wrist. A hollow wrist is essential when implementing the robot at Loogman, because it will prevent the vacuum hose from getting tangled with the robot arm. Replacing the IRB2400 by the Yaskawa Motoman GP25 will benefit this project because of two reasons:

- It is convenient to use the same robot in both the simulation and in reality.
- The Yaskawa GP25 is available at the TU Delft for further testing, making it accessible for this project.

In order to implement this arm in Godel, several packages have to be made or changed. To start, a macro.xacro file has to be created for the Yaskawa GP25 itself. The meshes of the robotarm are already available on GitHub [17]. Using these, a new macro.xacro can be made that will be compatible with Godel. By implementing this, a new xacro modelling the workspace has been created, keeping all existing blending tools, the modified democell and the new Yaskawa GP25 xacro. This democell can be visualised as a box around the setup, which constrains the arm from moving outside the work space. This cell had to be changed as well, since the larger GP25 did not fit into the original cell. This has been done by changing the dimensions of the democell.

To allow the arm to move in simulation, a MoveIt configuration package has to be built. This can be done by loading the workspace.xacro file into the MoveIt Setup Assistant. It is important to keep the same controllers and move groups as the original MoveIt package of the IRB2400. (Appendix B)

The next step is to change the Descartes files that plan the desired path of the simulated arm to the required location, so that the GP25 can move. First, a new Inverse Kinemetic Fast (IKFast) plugin has to be created for the GP25 because Descartes uses this algorithm for the motion planning of the robot. IKFast is used to transform a Cartesian path to joint poses for the robot. Inverse Kinematics differs from Forward Kinematics. Forward Kinematics determines the position of the tool tip on the angles of the joints of the robot. IKFast plans the angles of the joints based on the desired position of the tool tip [3]. To create this package the program OpenRAVE was used. Second, the Descartes files of the robot have to be modified, so that the new IKFast plugin can be used. (Appendix C to E)

C. Inserting the vacuum tool

Next the blending tool has to be replaced by the vacuum tool in the xacro file. The tool centre point (tcp)-frame, the frame that models the tip of the blending tool, has to be modified, so that it is positioned at the tip of the vacuum tool. Also the radius of the tool tip has to be adjusted to the size of the vacuum tool, this can be done in the configuration file path_planning.yaml. This is needed because the radius of the vacuum tool is bigger than the default tool size of Godel. (Appendix F)

D. Removing the Keyence laser scanner and the edge path

The Keyence laser scanner is used in Godel to check whether the surface has sufficiently been blended by the robot. It is a way to minimise errors in the quality of the blending process. This specific component of the Godel package, created for the blending robots, is not necessary for the vacuuming robot. After vacuuming it is not required to have a quality check, because this can simply be done by an employee. This will make the cleaning process more time efficient. In order to remove the Keyence laser scanner from the process a couple of files had to be modified.

Besides removing the Keyence laser scanner, the paths generated for both the scanner and the blending of the edges have to be removed too. This has been done by removing the generation of these paths in the blending_service_path_generation.cpp file. (Appendix G)

E. Optimising the path

A new path needs to be created for the vacuum cleaning to make it more efficient. The tool tip of the vacuum, provided by Loogman, is not circular but oval, giving issues when executing a path on a fixed angle, as is demonstrated in figure 1. Furthermore, Godel is based on a circular tool, which means that the program does not take the direction the tool is facing into account. When replacing the original blending tool with the vacuum tool, the direction the vacuum tool is pointing towards, has also been changed. This measure makes sure that the complete trunk can be vacuumed more efficiently, without the robot accidentally missing a spot.

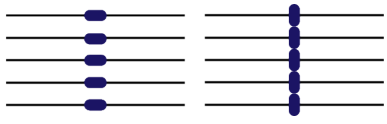

Fig. 1. On the left the wrong tool position causes gaps between the paths, on the right the correct tool position covers the whole height of the surface.

The default tool path supplied by Godel uses the spiral pattern shown in the third picture in figure 2, where the

mentioned issues are present. To solve these issues three potential solutions have been researched. These solutions were all developed without redesigning the whole path planning algorithm. The first solution is to increase the density of the path, and in that way covering more ground. The second one is, using the zigzagging path of the Keyence scanner, covering the whole surface with only horizontal movement. The last one is actively changing the angle of the vacuum tool to always be perpendicular to the path. These three solutions are shown in figure 2.

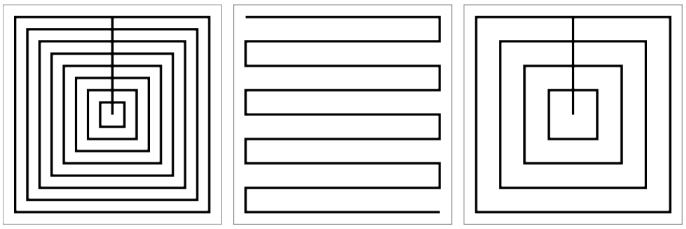

Fig. 2. Three ways to vacuum, from left to right: increased density, zigzagging (only horizontal movement), actively changing tool angle (covering the whole surface).

To find the best solution, the lengths of the paths have been compared to find the shortest path. With solution three being the default length. The two different paths were compared to the default length to calculate the difference in path length. For these calculations, parameters such as the speed and agility of the robot have not been taken into account.

Calculation of the path length, with n being the unit length of the side of the square shaped surfaces:

$$Increased\_density = 4\frac{n(n+1)}{2} + \frac{n}{2} = 2n^2 + \frac{5}{2}n \quad (1)$$

$$Zigzagging = n(n+1) + n = n(n+2) \quad (2)$$

$$Default\_and\_Rotating = 4\frac{n(\frac{n}{2}+1)}{2} + \frac{n}{2} = n^2 + \frac{5}{2}n \quad (3)$$

Efficiency calculations:

$$\frac{Zigzagging}{Default} = \frac{2(n+2)}{2n+5} \quad (4)$$

$$\frac{Increaseddensity}{Default} = \frac{4n+5}{2n+5} \quad (5)$$

Comparing the lengths of the paths:

This table shows the different lengths with respect to the values of n, with n=10 being the average value for the planned paths for the car trunk.

TABLE I
LENGTH OF THE PATHS AS A PERCENTAGE OF THE DEFAULT PATH

	n=0	n=1	n=10	$n = \infty$
Zigzag	80%	86%	96%	100%
Increased density	100%	128%	180%	200%

This research concludes that the zigzagging path is 4% more efficient than the default path. Making it the selected path for the vacuuming of the trunk. Implementing the path is done by replacing the default spiralling path, with the zigzagging path generated by the Keyence scanner. This has been done by editing the blend_planner.cpp file. This file calls the blend path generation service and then publishes poses as blend poses. The file has been changed in such a way that it calls the path generator package of the scanner (zigzag pattern) and then publishes the poses as blend poses instead of scan poses. The default spiralling tool path and the Keyence scanner path are shown in figure 3, as well as the new blending path in figure 4. (Appendix H,I)

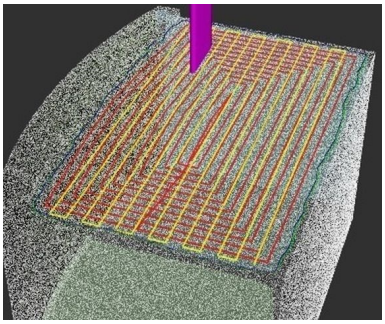

Fig. 3. Point cloud of the trunk with the default path in red, and the Keyence scanner path in yellow.

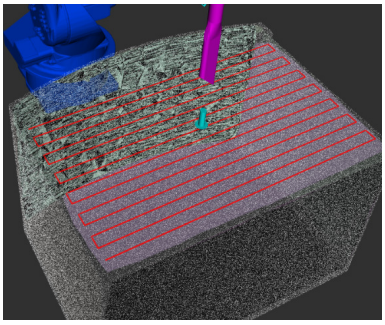

Fig. 4. Point cloud of the trunk with the new blending path.

F. Implementing the collision avoidance

One of the challenges is the implementation of collision avoidance. The robot has to recognise all parts of the trunk and make sure it does not collide with them. A solution to this is implementing Octomap. This has been done by using a Godel fork created by G.A.vd Hoorn [8]. This way the Descartes path planner avoids collision by checking the collision for each tool position, and changing it according to figure 5.

However, when using this software, only the spiralling tool path can be used. This is because of how the path is made; when creating the spiralling tool path, the blend path generation service updates the Descartes regularly, thereby taking collisions into account. The zigzagging path does not do this, because the method of collision checking is not yet compatible with the new path definition. (Appendix J)

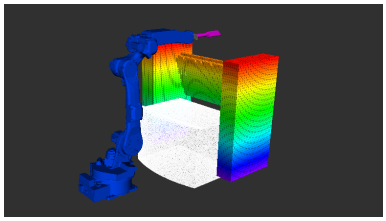

Fig. 5. Point cloud of the trunk with voxels representing the rest of the car

G. Demonstration

Initially, the Yaskawa GP25 was implemented in Godel for the simulation. However, the Yaskawa GP25 was not available anymore. Therefore the ABB IRB1200 was chosen for the demonstration, since it was available and already incorporated in Godel. This robot does not, in contrast to the GP25, have a hollow wrist. This means that a full vacuuming demonstration with this robot is not possible, because the vacuum hose might become entangled with the robot arm. The robot can still execute the planned path though.

A problem occurred while trying to get the real robot working. Godel did generate the path and planned the movements, but the robot did not execute them. In order to fix this problem, changes were made to the irb1200_blending.launch file and the moveit_planning_execution.launch file. This allowed Godel to send commands to the controller and for the robot to execute them.

However, the robot would only execute half of the planned path, due to a data limit on the robot controller. The robot controller could only execute a maximum trajectory length of one hundred points. The generated path consisted of significantly more points and this caused the robot to stop executing after it reached its maximum. In order to fix this problem, the maximum trajectory length had to be increased. This was done by changing the limit on the controller of the robot. After changing this value, the controller had to be restarted with the so-called 'P-start' (RAPID restart). (Appendix K)

V. Results

After choosing Godel as the software that is going to be used, the following features are achieved to complete the goal of the research:

- A new arm is implemented in Godel, because it is more suitable for the goal of automating the vacuuming process. This result is achieved by writing a xacro of the Yaskawa GP25 and by creating the required objects and models.
- A new MoveIt configuration is generated with the MoveIt setup assistant to make it possible for the robot to move in the simulation.
- An IKFast package is created to make the planning part faster.
- The vacuum tool is modelled and implemented in the simulation.
- Godel scans the pre-defined point cloud data and visualises the detected surfaces. Then it gives the option to select which surface Godel needs to plan a path for.
- Godel only plans a blend path.
- The blend path has been changed from a spiralling to a zigzagging path.
- The maximum trajectory length on the controller of the IRB1200 is changed to enable the robot to execute the whole path instead of just a segment.
- Collision avoidance is implemented to make it possible for the robot to avoid the walls of the car trunk.

VI. Discussion

The results show that the floor of a car trunk can be cleaned. However, there are certain aspects of the project that need thorough attention before implementation can start.

First of all, when using the zigzagging scan path, the path is generated using approximations. Most trunks have small to big curvatures on at least one side. The current scan path does not take small curvatures into account and will treat the surface as if it were rectangular. This can be optimised, in order to clean the trunk floor entirely.

Second, only three tool paths have been analysed in this project, due to their availability and easy implementation. However, there are infinite possible tool paths and other paths might be better suited for this application than the current paths. Also when finding the best path, parameters such as speed and agility of the robot have not been taken into account, even though these could significantly change the outcome.

Third, the robot that was initially chosen might not be the best suited robot for this project. The Yaskawa GP25 was chosen because of its hollow wrist and availability. However, when implementing a robot these are invalid arguments. There is no Motoman driver in Godel yet, since the implementation was more difficult than expected. Extended research should be done on focusing what robot is most suited for this application.

Finally, the absence of the scan part of the Scan-N-Plan leaves room for improvement. To fully use the flexibility of Godel, the scanning part is crucial.

VII. Conclusion

The goal of this project was to perform a demo of a robot arm automatically vacuuming the floor of a car trunk, showing the feasibility of the project.

A working simulation with the Yaskawa GP25 has been created. Godel starts with scanning the environment and is able to detect the given point cloud of the car trunk. It is also possible to choose which of the detected surfaces Godel has to plan a path for. On the selected surface a zigzagging blend path will be generated, covering the chosen surface. The following step is to execute, which is done in simulation by the virtual robot, but also by the real robot if it is connected. In the demonstration the ABB IRB1200, executes a zigzagging path in order to vacuum the mock-up of the trunk.

To avoid colliding with the car trunk Octomap was used in the simulation. This only works with the spiralling path, but

it creates a working collision 3D grid mesh, resulting in the robot avoiding the walls of the car trunk when planning the joint trajectory path.

To conclude this research project has contributed to the project of designing a fully automated car interior cleaning system, by creating a simulation and a demonstration in which the trunk of a car is being vacuumed by a robot. This shows that automatic car interior cleaning is a feasible option for Loogman to use in the future.

VIII. Recommendations Short-term

Before the robot can be fully implemented in the interior cleaning process, more research has to be done. This will take a lot of time and therefore it is a long-term solution. When looking for a short-term solution, it might be better to look for other possibilities. For example, hybrid robotic assistance systems that are used in the car manufacturing industry could be redesigned for application at Loogman. Another option is a small vacuum robot, which is put in the trunk by employees and vacuums the trunk autonomously. This is easier to implement in the interior cleaning process at this moment. However, these solutions do not offer the multi-employability of generic robot arms. Researching more basic solutions will yield more returns in the short term.

IX. Recommendations Long-term

This project is a good start in realising the full automation of the interior car cleaning process at Loogman. Suggested next steps are the optimisation of this project, as well as looking into developing a fully automated cleaning system.

To finish the goal of full automation it is recommended to combine the current issues mentioned in the discussion with the following recommendations, to create a working robot with the optimal path and a working collision avoidance.

At the moment, the software needs two different point cloud inputs to generate both the collision meshes with Octomap and the surfaces that need to be tooled. It would make more sense to give one point cloud as input and automatically select the surfaces to be tooled. Once a surface is selected, all the other surfaces should be converted into collision meshes, thereby erasing the need for multiple point clouds. This could be implemented in the future.

In the current project the scan part of the Scan-N-Plan method was skipped. The robot arm is supposed to scan the trunk by itself and generate the required models, instead of receiving pre-programmed data as described in this paper. Receiving these pre-made models limits the robot to only vacuuming the given trunk, and thus withholds the robot from being able to vacuum the trunk of any car. It is necessary to get the scan part working before the robot can be implemented in the process. Therefore it is recommended to do more research into the scan part of Scan-N-Plan.

To optimise the vacuuming, it is useful to develop a new vacuum tool specialised for the application of vacuuming. Humans can adjust their force manually while vacuuming, for example when the car is moving a bit. To prevent the car from getting damaged due to too much pressure from the robot, a new vacuum tool could be designed. Instead of a new vacuum tool that regulates the applied force, it is also possible to let the robot control the force applied. The version of Godel used in this project does not have this feature, so this needs be added. Furthermore, the shape of the tool could be optimised for different tool paths, improving the efficiency of the process.

Finally, the implementation of a fully automatic car cleaning system is something that requires research on its own. The interior car cleaning process contains different types of tasks which will need multiple robots. Looking into automating the whole process with robots requires a lot of work in itself.

X. References

1) ABB Experimental package, Consulted in December 2019. `https://github.com/ros-industrial/abb_experimental`
2) Bezier package, Consulted in October 2019. `https://github.com/ros-industrial-consortium/bezier`
3) S. Chitta, D. Hershberger, A. Pooley, D. Coleman, M. Gornera, F. Suarez, and M. Lautman, *MoveIt! tutorials: IKFast Kinematics Solver*, Consulted in October 2019
4) R. Diankov, *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010. `http://www.programmingvision.com/rosen_diankov_thesis.pdf`
5) Ensenso GmbH, *About us*, Consulted in December 2019. `https://www.ensenso.com/about-us/`
6) D. Girardeau-Montaut, *Cloudcompare*, Consulted in December 2019. `https://www.danielgm.net/cc/`
7) Godel package, Consulted in October 2019. `https://github.com/ros-industrial-consortiumconsortium/godel`
8) G. vd Hoorn, *GitHub repository Octomap*, Consulted in December 2019. `https://github.com/gavanderhoorn/godel/tree/add_octomapmoveit_descartes`
9) R. Madaan, *Descartes*. ROS.org, Consulted in December 2019. `http://wiki.ros.org/descartes`
10) MoveIt, *MoveIt Documentation*, Consulted in December 2019. `https://moveit.ros.org/documentation/`
11) ROS, *About ROS*, Consulted in September 2019. `https://www.ros.org/about-ros/`
12) ROS, *Ubuntu install for the use of ROS Kinetic*, Consulted in September 2019. `http://wiki.ros.org/kinetic/Installation/Ubuntu`

13) ROSIN Project consortium, *ROS-I Academy Training*. https://tudelftroboticsinstitute.nl/study/ros-academy
14) ROS-Industrial consortium, *Scan-N-Plan*, Consulted in September 2019 https://rosindustrial.org/scan-n-plan
15) SoloLearn, *C++ Tutorial*. https://www.sololearn.com/
16) K. Wurm, *Octomap*. ROS.org, Consulted in December 2019. http://wiki.ros.org/octomap
17) Yaskawa GP25 support package, Consulted in October 2019. https://github.com/fizyr/yaskawa_gp25_support
18) Yaskawa Motoman, *Handling & General Applications with the GP-series*, Consulted in November 2019. https://www.yaskawa.eu.com/index.php?eID=dumpFile&t=f&f=17300&token=6b0b872f02729308f584c0daa937652f249879f4

XI. Appendix

A. Tutorials

To learn the basics of the software used in this project the following tutorials supplied for this project were used, recommended for beginners for working with ROS.

- Installing Ubuntu for ROS [12]
- ROS-I Academy Training [13]
- Learning C++ [15]

B. MoveIt configuration adjustments

In order to make it possible for the Yaskawa GP25 to move during simulation, a new MoveIt configuration had to be made. To make it suitable for Godel the following files had to be created or adjusted:

- All links, joints and virtual joints in the GP25 MoveIt configuration have been given the same names as used in the IRB2400 files, since Godel uses these names in the remaining parts of the program.
- The fake_controllers.yaml file has to be exactly the same as the one used for the IRB2400, because these specific controllers are called upon by the rest of the Godel program.
- In the ompl.yaml file, all the default values created by the MoveIt setup assistant were put to none. These had to be removed.
- The ros_controllers.yaml file is called upon by the GP25 MoveIt configuration, while the IRB2400 configuration calls upon a controllers.yaml file. The solution to this was adding the controller mentioned in the controllers.yaml to the ros_controller.yaml.

C. Creating IKFast

OpenRAVE is a special software developed to create IKFast solutions for robot arms. These solutions are incredibly powerful and fast in calculating the possible positions of the robot arm. In order to make the files that create these solutions, OpenRAVE needs a collada file as input. ROS can automatically create these files from URDF files with a simple command. An URDF file of the setup can be easily created by loading the workspace.macro.xacro into the MoveIt Setup Assistant. This .xacro file has been created and converted to the collada format. Next OpenRAVE created the needed IKFast files.

D. First implementation of the IKFast

The original Godel structure has an IKFast package containing two .hpp files and one plugin_init.cpp file. However, the IKFast created by OpenRAVE has two .cpp files that have almost the exact same content as the two .hpp files. In order for Godel to work, the newly created IKFast files need to have the same structure. This has been achieved by copying the existing files and their structure and changing them to the new GP25.

The first step was to add the .hpp files in the IKFast manipulator package and copy and paste the contents of the GP25 IKFast .cpp files into them. Next the names of these files had to be changed to make sure that they are specific for the GP25. In the new GP25 IKFast MoveIt plugin .hpp file, the IKFast solver should call the .hpp file instead of the original .cpp file. Next, the plugin_init.cpp file had to be changed, making sure it would call upon the right .hpp files as well as changing the pluginlib export class from irb2400_ikfast_manipulator_plugin to ikfast_kinematics_plugin. Since this is the class that is created by the GP25 IKFast files. Also the CMakeLists.txt file had to be changed: In the add libraries section, now the gp25_plugin_init.cpp has to be called instead of the old .cpp file.

E. Creating a new Descartes package for the GP25

After copying the Descartes package in the Godel IRB2400 folder, the first thing that has to be changed is the CMakelists.txt file. The original file referred to the IKFast package of the IRB2400, while it should now refer to the GP25 IKFast package. Next, the plugins.xml file has to be adjusted, renaming the class and the type of the robot model, that will be called upon from the gp25_blending.launch file. Finally the package.xml file has to be changed so it refers to the GP25 IKFast package too.

Now that the Descartes package is compatible with the IKFast package of the GP25, the files inside of the package can be changed, starting with the robot_model.h file and with the kinematics plugin. These have to be changed just as mentioned before in the implementation of the IKFast section. Furthermore all the classes that are defined using the ABB IRB2400 have to be replaced with the GP25 naming.

Next the robot_model.cpp file has to be changed as well. For this file the same changes are needed as for the previous file: changing the kinematics type as well as replacing IRB2400 with GP25.

F. Adding the vacuum tool

This can be done by replacing the blending tool in the xacro to the tip of the vacuum tool. Afterwards the tcp-frame has to be changed so it is positioned at the tip of the new tool. Next the blend_process_planning.cpp has to be changed so it positions the tool in the right direction. This has been done by modifying the toDescartesBlendPt at line 34, so it only accepts the tool in special poses, which helps the cleaning go smoother.

G. Deleting the Keyence sensor

The Keyence laser scanner is not necessary in this project so it has been deleted completely. To delete the whole scanner all the files have to be checked for a scanner part. This can be done by looking out and searching for the scanner in all the files. In the end the scanner has been deleted from the following files:

- Surface_blending_service.h
- Surface_blending_service.cpp
- Blending_service_path_generation.cpp
- Blending_eff_macro.xacro
- Godel_process_planning_node.cpp
- Godel_process_planning.h
- Godel_process_planning.cpp
- Process_planning.launch
- Gp25_blending.launch

For removing the scan path and the edge path, lines 272 to 314 have been removed in the blending_service_path_generation.cpp file.

H. Adjusting the path

There are two files where the paths for the blending and for the scanning are defined, respectively in blend_planner.cpp and scan_planner.cpp. The part of the script of scan_planner.cpp where the path is described was copied and pasted in blend_planner.cpp, replacing the original blending path with the scan path. Besides this, the growth factor in profilometer_scan.cpp was changed as well. The growth factor is a scaling factor which scales the path. The growth factor had to be decreased to acquire the desired coverage of the surface of the trunk with the planned path. At last, the file profilometer_scan.h had to be called in blend_planner.cpp in order to be able to run.

I. Adjusting the speed in simulation

The robot was moving slow while in simulation, so the speed of the robot had to be adjusted. In godel_process_planning/src/common_utils.cpp the default values are set, including the “DEFAULT_JOINT_VELOCITY” in line 25. This initial value was of the IRB2400, so it had to be changed to the joint velocity limit of the Yaskawa GP25. This limit is 210 degrees/s, which is equal to 3.665 rad/s [18]. Besides this value, the “DEFAULT_MOVEIT_VELOCITY_SCALING” in line 30 had to be changed. The value was initially set to 0.1 to slow down the movements of the robot. But because the robot had to move faster, this value was changed to 1.0.

J. Implementing the collision avoidance

To achieve the implementation of the collision detection certain files had to be changed according to the Godel fork that G.A. vd Hoorn posted on GitHub. This version of Godel added a planning scene to Descartes, thereby making it compatible with the newly produced data by Octomap. Also it updated the path planning files, to make sure the Descartes would take Octomap into consideration. Next a new Octomap sensor was added, that could read point cloud data and convert it to voxels. To get this sensor working with the current program, a new node was written in the gp25_blending.launch file. This node reads point cloud data (PCD) and publishes it on a topic that the new sensor reads. This sensor can then change the data so it will be regarded as collision data.

K. Getting the real robot working

The abb_driver already provided by Godel was not sufficient to make the real robot work so the abb_irb1200_support package from the abb_driver had to be switched out for the new abb_irb1200_support package from the new abb_experimental package. The abb_experimental driver package can be found on GitHub [1]. After downloading the abb_experimental package some changes had to be made in the moveit_planning_execution.launch file of the IRB1200. In line 6 the joint_names_irb1200.yaml had to be altered to joint_names_irb1200_5_90.yaml. Also lines 34 t/m 53 had to be removed and replaced by:

<group unless = ” $(arg sim)”/>
<include file = ” $(find abb_irb1200_support/launch/robot
_interface_download_irb1200_5_90.launch/>
<arg name = ”robot_ip” value = ”$(arg robot_ip)”/>
</include>
</group>

In irb1200_blending.launch in lines 32 and 36 the nodes had to be switched. So in line 32 the type = ”blend_process_service_node” had to be switched out with ”abb_blend_process_service_node”.

This solved the problem of connecting the real robot with the Godel program. However, the controller could only accept a maximum of 100 points while Godel would send more, which resulted in that the robot would only execute a fraction of the trajectory. This has been solved by updating the amount of points the controller can receive. First of all the ROS_commons.sys file in Godel had to be changed; the value of the MAX_TRAJ_LENGTH had to be adjusted to 500. That way Godel was able to send all the necessary points to the controller. Next, the trajectory length had to be changed on the controller itself, so that it accepts a maximum of 500 points. This has been done via ABB studio. In RAPID the ROS_commons.sys file has been changed in the same way as

in Godel. Finally, the controller had to be restarted to make sure that these changes were saved using a P-start.