



## FACULTY OF SCIENCE AND TECHNOLOGY

# MASTER'S THESIS

Study programme/specialisation: Automation and signal processing	Spring / Autumn semester, 2018.  Open/ <del>Confidential</del>
Author: Eivind Sandve Haus	 (signature of author)
Programme coordinator: Morten Mossige	
Supervisor(s): Andreas Skaar	
Title of master's thesis:  The use of motion tracking for teaching of paint robots  Norwegian title: Bruk av bevegelsessporing for opplæring av lakkeringsroboter	
Credits: 30 ECTS	
Keywords:  ABB, paint robots, HTC Vive, motion tracking, robot teaching, automated painting	Number of pages: 60  + supplemental material/other: <a href="#">source code</a>  Stavanger, 15.06.2018 date/year

## Abstract

Most industrial production processes require parts to be painted, both for aesthetic and quality purposes. The painting process is often automated with the benefits of better paint coverage, less waste of paint, high consistency between batches and reduced production time. One way to automate this process is to mount a spray-painting gun on an industrial robot. The robot then moves along a preprogrammed path and covers the object in paint.

Before it can be used the robot must be programmed. This consists of determining the path the robot should follow and when it should spray paint. A new paint program must be made every time the product is changed or a new part needs to be painted. This project has developed an intuitive and efficient way of creating said paint programs.

The approach is to use the fluid motion of the human arm and experience of painters to create the robot's path. This is performed with the use of motion tracking technology. The operator holds a hand held device which is tracked in three-dimensional space. When the operator moves the device as if painting the object, the traced path is recorded and used to create the paint program. The virtual reality system HTC Vive has been used for the motion tracking. It has shown promising results in precision and reliability in the past, which have been confirmed by this project.

The project has implemented an interface to retrieve information from the HTC Vive system. Two approaches to the paint program creation has been developed. The first approach records the path independently of the robot and creates the paint program. It is then loaded onto the robot and can be used to paint the object. The second approach uses motion tracking to control the robot in real-time. While the operator controls the robot the object is painted and the path is recorded. The path is then used to create the paint program which can be loaded onto the robot to paint new objects.

Both of the implemented solutions have shown promising results. They allow the user to create simple paint programs in a matter of minutes with little to no training needed. There are some limitations when creating more advanced paint programs, especially in regards to determining the correct orientation of the paint gun tool. More work would be necessary to make the implemented solutions ready to be used commercially.

# Contents

<b>List of Figures</b>	<b>4</b>
<b>Code excerpts</b>	<b>5</b>
<b>1 Introduction</b>	<b>6</b>
1.1 Industrial painting . . . . .	6
1.2 Automation of spray painting . . . . .	6
1.3 This project's idea . . . . .	7
1.4 Defining the tasks . . . . .	7
1.5 Structure of the report . . . . .	7
<b>2 Prerequisites</b>	<b>9</b>
2.1 Motion tracking . . . . .	9
2.1.1 HTC Vive . . . . .	9
2.1.2 OpenVR API . . . . .	11
2.2 Robotics . . . . .	13
2.2.1 3D transformations . . . . .	13
2.2.2 ABB robots . . . . .	15
2.2.3 Externally Guided Motion - EGM . . . . .	16
2.2.4 Simplified Robot Programming - SRP . . . . .	17
2.3 Programming tools . . . . .	18
<b>3 Implementation</b>	<b>20</b>
3.1 Hardware . . . . .	20
3.2 Software . . . . .	21
3.3 ViveTracker . . . . .	21
3.4 Simplified Robot Programming . . . . .	23
3.4.1 System overview . . . . .	23
3.4.2 Implementing new sensor . . . . .	24
3.4.3 New recording mode . . . . .	25
3.4.4 Reachability checking . . . . .	25
3.5 Online Teaching . . . . .	28
3.5.1 System overview . . . . .	28
3.5.2 Robot program . . . . .	29
3.5.3 PC application . . . . .	32
3.5.4 Continuous mode . . . . .	34
3.5.5 Point to point mode . . . . .	37

<b>4 Tests and results</b>	<b>40</b>
4.1 Simplified Robot Programming . . . . .	40
4.1.1 Continuous recording . . . . .	41
4.1.2 Point to point recording . . . . .	42
4.1.3 Reachability checker . . . . .	43
4.2 Online Teaching . . . . .	43
4.2.1 Continuous mode . . . . .	44
4.2.2 Point to point mode . . . . .	45
<b>5 Discussion</b>	<b>46</b>
5.1 Hardware . . . . .	46
5.1.1 Sensor system . . . . .	46
5.1.2 Controller . . . . .	46
5.2 ViveTracker interface . . . . .	48
5.3 Simplified Robot Programming . . . . .	48
5.3.1 Recording mode . . . . .	49
5.3.2 Reachability checking . . . . .	49
5.3.3 Improvements and future work . . . . .	51
5.4 Online Teaching . . . . .	51
5.4.1 Calibration and recording . . . . .	52
5.4.2 Possible solutions to singularity problems . . . . .	54
5.4.3 Improvements and future work . . . . .	55
<b>6 Conclusion</b>	<b>56</b>
<b>7 References</b>	<b>57</b>
<b>Appendices</b>	<b>60</b>
<b>A Online Teaching source code</b>	<b>60</b>

# List of Figures

1	HTC Vive - ©2016 BagoGames, license (CC BY 2.0) . . . . .	10
2	HTC Vive room setup - ©HTC Corporation, with permission . . . . .	11
3	HTC Vive Tracker - ©HTC Corporation, with permission . . . . .	11
4	IRB52 - ©ABB AB Robotics, with permission . . . . .	15
5	EGM motion - ©ABB AB Robotics, with permission . . . . .	17
6	EGM UDP interface - ©ABB AB Robotics, with permission . . . . .	17
7	Simplified Robot Programming - ©ABB AB Robotics, with permission . . . . .	18
8	Vive controller - ©HTC Corporation, with permission . . . . .	22
9	SRP system overview . . . . .	23
10	SRP GUI - ©ABB AB Robotics, with permission . . . . .	24
11	SRP settings - ©ABB AB Robotics, with permission . . . . .	25
12	SRP create reachability checker - ©ABB AB Robotics, with permission . . . . .	26
13	IRB52 working area - ©ABB AB Robotics, with permission . . . . .	27
14	Reachability checker - ©ABB AB Robotics, with permission . . . . .	27
15	Online Teaching system overview . . . . .	29
16	Configuring UDP - ©ABB AB Robotics, with permission . . . . .	29
17	Configuring EGM - ©ABB AB Robotics, with permission . . . . .	30
18	Signals - ©ABB AB Robotics, with permission . . . . .	30
19	Online Teaching - Menu . . . . .	33
20	Online Teaching - Netscan . . . . .	33
21	Online Teaching - Work object . . . . .	34
22	Online Teaching - Continuous mode . . . . .	35
24	Online Teaching - Point to point mode . . . . .	37
23	Online Teaching - sequence diagram . . . . .	39
25	Recorded path - Continuous flat . . . . .	41
26	Recorded path - Continuous object . . . . .	41
27	Recorded path - P2P flat . . . . .	42
28	Recorded path - P2P object . . . . .	42
29	Reachability checker test- ©ABB AB Robotics, with permission . . . . .	43
30	Recorded path - Continuous . . . . .	44
31	Recorded path - P2P . . . . .	45

# Code excerpts

1	ViveTracker UpdateData . . . . .	21
2	Haptic pulse . . . . .	22
3	Online Teaching RAPID EGMRun . . . . .	32
4	Online Teaching work object . . . . .	34
5	Online Teaching calibration . . . . .	36
6	Online Teaching sent data . . . . .	36
7	RAPID - DPad Trap routine . . . . .	38

# 1 Introduction

## 1.1 Industrial painting

Painting parts is an integral element of most industrial production processes. It can be important both for aesthetic purposes and to prolong the lifetime of the product. Covering an object in paint can be done in several ways. Possible applications include dipping the object in paint, spray painting or direct application with a roller or brush. The process of spray painting will be the focus in this project.

Spray painting uses a tool or "gun" that through compressed air or other means propels the paint towards the surface to be covered. The painter moves the gun to cover the object satisfactorily. This is tedious and repetitive work and generally very time consuming for humans to perform. Fumes from the paint are also often detrimental to human health, making protective equipment and precautions necessary. These are some of the reasons why spray painting is often automated.

## 1.2 Automation of spray painting

One way to automate the spray painting process is to mount the paint gun on an industrial robot. The robot then moves along a preprogrammed path and sprays paint to cover the object. The object can both be fixed in place or moving on a conveyor. In general an automated system will give better and more even coverage of the object with less waste of paint. It also ensures that the result is the same every time. Painting can also be done much faster and an automated system can decrease both production time and cost. All this depends on the robot being properly programmed.

Programming a paint robot is often called teaching and consists of generating the path the robot should follow and determining when the paint gun should be turned on and off. There are several ways to program a robot, but they are in general time-consuming and need highly skilled programmers. In many cases it might be necessary with external consultants when paint programs for new parts need to be made, which increases the time usage and costs of making changes to the production process. When high precision is needed or when the production volume is high, the time and money spent on optimizing the automated painting is offset by the high gain in efficiency. For smaller volumes, however, the long time spent automating the process might not be worth it compared to spray painting it manually.

## 1.3 This project's idea

The motivation behind this project is to make the process of teaching paint robots simpler and more efficient. The idea is to convert the fluid motion of the human arm and the prowess of experienced spray painters into robot programs. With the use of motion tracking technology the operator's hand movement can be recorded and used to create the paint program. This is a very intuitive way of teaching the robot which requires very little training and no in-depth knowledge about robots. The time saving potential of such a system is also large. This project will seek to develop such a system.

## 1.4 Defining the tasks

The project is in cooperation with ABB Robotics, Bryne. It is divided into two parts that have approximately the same amount of focus. The two tasks show slightly different approaches to solving the same problem.

ABB has an existing system called Simplified Robot Programming that uses external motion tracking to teach paint robots. The sensor used in this system has some restrictions that severely hinders the usability. The first task consists of expanding this system to use a new sensor with better capabilities. It also involves adding new functionality to the system to increase its merit for teaching paint robots.

The second task builds on a project conducted at University of Stavanger in fall of 2017 [6]. This used motion tracking of a hand held device to control an ABB robot in real-time. The task is to create a system where the paint robot moves and paints in real-time while the path is being recorded. Similarly to the first task, functionality to make it as user friendly as possible should be added.

For both tasks the output from the system must be on a form that can be run on ABB's paint robots. It must also be able to use existing software for displaying and editing the programmed paths. The project will only focus on the creation of the robot paths. Other aspects of automated painting such as the tools used will not be touched upon.

## 1.5 Structure of the report

This report consists of six chapters. Chapter 1 is this introduction with background and task definitions. The next chapter will cover prerequisites that may be necessary to read the report. This includes some theory about robotics and an introduction to existing solutions used in the project. Chapter 3 covers the

implementation of the created systems. In chapter 4 tests of the different solutions are presented. Chapter 5 discusses the different solutions, highlighting challenges and possible improvements that could be made. The last chapter presents the conclusion of the project and to what degree the presented tasks have been solved.

## 2 Prerequisites

This section presents some subjects the reader should be familiar with before reading the rest of the report. It includes technologies and solutions that the systems created in the project are dependent on.

### 2.1 Motion tracking

Tracking is in this context understood as determining an objects position as a function of time. For an object in 3D this involves finding both its position and orientation, often called the pose of the object. This position will be relative to a reference frame defined by the system. There are many ways of approaching the problem and examples of systems using cameras, infrared lights or physical cables to find the position exists.

To be usable in conjunction with robots such a system needs high precision and good reliability. Such systems for the corporate market have existed for a while, but often sports a high price tag. The recent surge in popularity of Virtual Reality (VR) systems is bringing the possibility of high quality motion tracking to a much broader market.

#### Virtual Reality

Virtual Reality systems lets the user step into a computer generated 3D environment. The user wears a headset with a stereoscopic display that shows the virtual world. The level of immersion of such a device is far greater than watching on a screen. If the system includes tracking of the headset's position, the user can move and look around in the virtual world. Through the use of hand held controllers the user can also interact with the virtual environment. To achieve this the position and orientation of all these devices in relation to each other and the play area must be known. Virtual reality started as a tool for interactive movies and games, but the high performance and relatively low cost of these systems are also making them popular for uses outside of entertainment.

##### 2.1.1 HTC Vive

HTC Vive is a VR system developed through a cooperation between HTC Corporation and Valve Corporation. It was released in 2016 and was created to be used with games and other virtual reality media. This section will explain the basis for the system's tracking and key capabilities of the system as a sensor.

The HTC Vive set consists of the three main parts seen in figure 1. The HMD is worn on the user’s head and provides a stereoscopic display of a virtual environment. The two hand controllers are used to interact with the environment through several buttons on the controllers. The Lighthouses are fixed base stations mounted in the room. They are vital to the tracking performance of the system.

## How tracking is performed

To track the devices the system combines two methods. The pose of each device is found relative to each other and to a fixed reference frame defined by the system.

The first method is the use of an inertial measurement unit (IMU). This is a combination of an accelerometer and a gyroscope which measures the change in acceleration and orientation. Through integration it can find the position as it changes over time. The device does not know its absolute position in 3D space, but only where it has moved relative to where it started. Inaccuracies in the measurements will accumulate over time and cause increasing error from its actual position. To correct this error the device’s position is periodically updated from a stationary reference, the Lighthouses.

The Lighthouses alternately scan the room with two infrared laser lines, horizontal and vertical. This infrared light is used by the devices to update their current pose and correct the estimate found by the IMU. Each device has a number of optical sensors distributed around the device. The IR lasers from the Lighthouses will hit these optical sensors at different times depending on the device’s orientation. These time differences are then used to calculate the absolute pose of the device. When two Lighthouses are used they are synchronized through a flash before each scan starts. It is possible to use only one Lighthouse with a reduction in tracking precision. Figure 2 shows how the devices work along with the Lighthouses to facilitate tracking.

Some key capabilities of the HTC Vive tracking system are listed below:[18]

- Provides position and orientation data for each tracked device (6 DOF).
- Refresh rate 250Hz.
- Jitter of 0.3 mm, up to 2.1mm if only one lighthouse is used.
- Expected practical precision around 2 mm.



Figure 1: HTC Vive set with the Head Mounted Display (HMD), two controllers and two Lighthouses

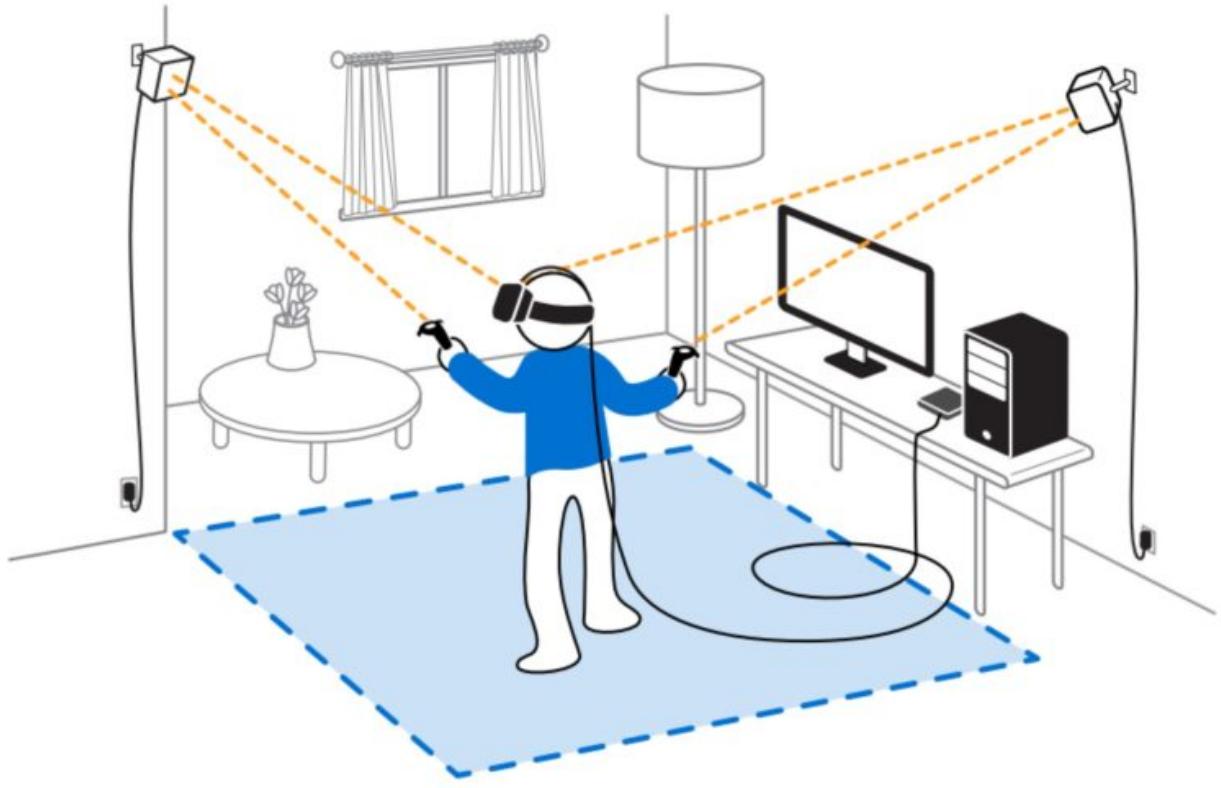


Figure 2: HTC Vive room setup. [16]

HTC have also made a smaller device called the Vive Tracker. It is an accessory meant to be mounted on devices to facilitate tracking and sports the same capabilities as the other tracked devices. It has a standard camera mount and a port for communication. Initially it targeted developers to be used in products but is now also available to the general public.



Figure 3: HTC Vive Tracker. [17]

### 2.1.2 OpenVR API

Along with the HTC Vive hardware Valve released an Application Program Interface (API), OpenVR API. This is an open source API that lets developers interact with different virtual reality hardware through the same programming interface. The API will detect what kind of hardware is being used and initializes the specific implementation. OpenVR can be run in different modes. The background mode where no rendering window for the display is required will be used in this project.

As openVR is a C/C++ library some adaptions must be made to be able to use it in C#. C++ creates unmanaged code which runs directly on the operating

system whereas C# creates managed code which runs in a Common Language Runtime. For previous projects ABB has implemented a C++/Common Language Interface as a bridge between the unmanaged OpenVR API and managed C#. This creates several layers of code between OpenVR and the C# API, increasing the complexity. To make changes it has to be propagated down through all the layers. Now the OpenVR API also includes a C# file which uses marshaling to bridge the gap between managed and unmanaged code. Functions can then be called directly in C# and makes development much simpler. This `openvr_api.cs` that implements the marshaling will be used in this project.

OpenVR has structures and functions that can be used to interact with the hardware. Those that are relevant to this project are described under:

- `OpenVR.Init(*)` initializes the connection to the hardware through SteamVR.
- `GetControllerState(*)` returns the state of the selected controller as a `VRControllerState_t`.
- `GetControllerStateWithPose(*)` returns the state but also includes the latest updated pose of the controller.
- `GetTrackedDeviceActivityLevel(*)` returns the activity level of the device. Can be used to check if the controller is ready and tracking.
- `TriggerHapticPulse(*)` triggers a haptic pulse on the controller causing it to vibrate.
- `VRControllerState_t` holds the state of the controller's inputs. This includes buttons, analog trigger and trackpad.
- `TrackedDevicePose_t` holds the information about a device's pose. It contains a 3x4 matrix with the pose and 3D vectors with velocity and angular velocity.

## SteamVR

The application using OpenVR does not connect to the hardware directly. This is handled by the SteamVR application. It connects to the HTC Vive devices and maintains their connection status. SteamVR has a room setup option where one can define the working area and the coordinate axes. When OpenVR is initialized it connects to SteamVR to retrieve information from the devices. As a consequence the SteamVR application must be running as long as the HTC Vive devices are being used. By default the SteamVR application requires the HMD to be present for it to run. For applications that only uses the controllers, this

can be a nuisance. It can be circumvented by changing the option "requireHmd" to false in the settings file for SteamVR.

## 2.2 Robotics

The field of robotics is vast and giving a full intro to the field in this report is not feasible. This section will give a very brief introduction to robots based on [1] and then focus on the aspects most important to this project.

A robot manipulator is a chain of links connected by joints. A joint is usually either revolute (rotating motion) or prismatic (linear motion). The relative displacement between two links is called the joint variable. Knowing all the joint variables the exact position of each point on the robot manipulator can be inferred. The point in interest is usually the end effector which is the end of the final link where a tool typically is mounted. The workspace of the robot is all the possible positions this end effector can reach given all possible combinations of joint variables. This is dependent on the geometry of the manipulator and possible constraints on the joints. An object in three dimensional space is described by six degrees of freedom (DOF), three for position and three for orientation. To be able to reach an arbitrary point with an arbitrary orientation a robot manipulator therefore needs six independent DOF, which is the reason most industrial robot arms possess at least six joints.

### 2.2.1 3D transformations

Each object in three-dimensional space has its own coordinate system. For some objects this is fixed like the base of a robot. Other objects like the end effector of the robot or a motion tracked device will move and the coordinate system moves with it. For several robots or systems to be able to interact they need to be able to describe their position in a common reference system. To facilitate this, 3D transformations are used. A 3D transformation consists of a translation and a rotation and describes one coordinate frame relative to another frame. One way of describing such a transformation is by use of homogeneous transformations ([1] chapter 2.7). This describes the transformation with a 4x4 matrix including both the translation and rotation part. Another way of describing such a transformation, which is much used when working with robots, is through the use of a translation vector and rotation quaternions.

### Quaternions

Quaternions are an extension of the complex numbers. A quaternion  $q$  is gen-

erally represented on the form:

$$a + bi + cj + dk$$

[3] where a, b, c and d are real numbers while i,j and k are the fundamental quaternion units. These quaternion units are connected by the formula

$$i^2 = j^2 = k^2 = ijk = -1$$

Other common notations for quaternions are  $q = W + Xi + Yj + Zk$  or  $q = q_0 + q_1i + q_2j + q_3k$ . The notation used by ABB is  $q_0-q_3$ . Quaternions have their own set of algebraic rules. Quaternion multiplication is for example not commutative, meaning that the order of the factors is not arbitrary:

$$q_1 * q_2 \neq q_2 * q_1$$

A unit quaternion is a quaternion with norm 1. These are also called versors, orientation- or rotation quaternions as they can be used to describe rotations in three dimensions. These are much used in robotics and 3D modeling because they are computationally efficient and interpolation between two orientations is simple. Another advantage is that they are not susceptible to "gimbal lock" which occurs when two axes aligns, resulting in loss of one degree of freedom.[4] Understanding the practical meaning of a quaternion can be hard for humans but it can be seen as similar to an axis/angle representation of rotations.  $q_1-q_3$  can be understood as a vector defining an arbitrary axis of rotation in three dimensions. The real part  $q_0$  then describes the angle of rotation around this axis. The internet site in [11] is a powerful tool that can be used to visualize quaternions and convert to Euler angles which are easier for humans to interpret. Some unit quaternion relations used in this project are given below.

Given a vector  $p$  and a quaternion  $q$  the resulting vector  $p'$  rotated by  $q$  is found from:

$$p' = q * p * q^{-1}$$

If given two orientations  $q_1$  and  $q_2$  as quaternions the relative rotation  $q_3$  from  $q_1$  to  $q_2$  can be found from:

$$q_3 = q_1^{-1} * q_2$$

An arbitrary number of rotation quaternions can be combined to form an equivalent rotation:

$$q_3 = q_2 * q_1$$

which is equivalent to a rotation of  $q_1$  followed by  $q_2$ .

## 2.2.2 ABB robots

ABB has a lineup of many different robots in its arsenal. The ones most commonly used for paint operations are six axis robots with all revolute joints like the IRB52 seen in figure 4. It has a base that is fixed to the ground/wall/ceiling and a tool mount at the end of the final link. Different versions of this kind of robot will be used in the project. For the robot to be able to operate it needs a controller. The controller handles communication, I/O-signals and many other important features. For paint robots the controller also incorporates the paint process control, IPS. The controller usually has some sort of teach-/flex pendant that lets the operator interact with it.

RobotStudio is the programming environment for ABB robots. It can be used both for offline programming and simulation and to remotely connect to real robots. The programming language used for ABB robots is called RAPID. It is a high level language specifically created to program robots. The language defines structures and routines to control program flow, robot movement, I/O signals and much more. Some important terms to know when working with ABB robots are explained below.

- The **Tool Center Point - TCP** of a robot is the point that will be moved to the target. By default this is the tool mount at the end effector. By adding a tool to the robot the TCP will move according to the geometry of the tool.
- **Work objects** are RAPID's way of defining different coordinate reference frames. If a set of targets are defined in one work object, they can all be moved together by moving the work object.
- A **robtarget** consists of a position and orientation and defines where the TCP should be if the robot moves to this target.

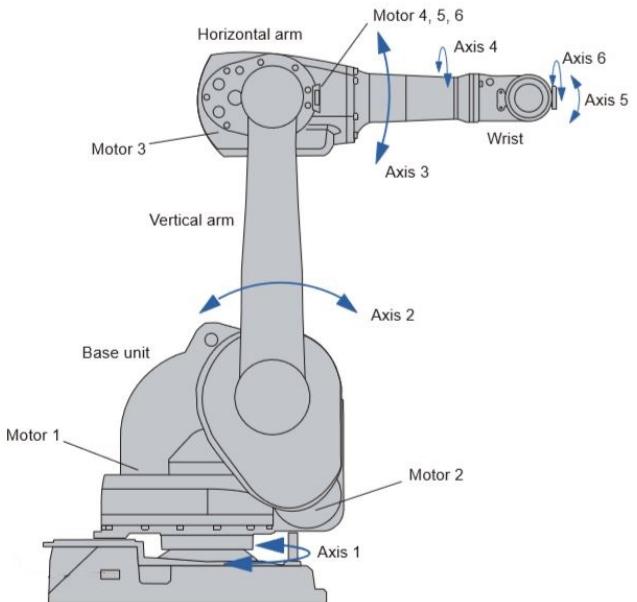


Figure 4: IRB52 with axes overview. [28]

- **Singularties** are points where the robot configuration to reach this point is not unique. The robot does not know which configuration to use and will generally lead to errors and program interruption.
- A **Brush** defines a set of parameters for the paint tool, effectively determining the shape and volume of paint being sprayed. It is defined in a brush table and the *SetBrush* command with the correct brush number is used to trigger paint on and off.

Two other ABB programs are used in this project. Robview is a PC tool for monitoring and controlling robot cells. ShopFloorEditor lets the user visualize and edit created paint programs.

### 2.2.3 Externally Guided Motion - EGM

Externally Guided Motion is a relatively new module for use with ABB robots. It offers three different features:

- EGM Position Stream - Positions of mechanical units in RAPID tasks are sent to external equipment.
- EGM Position Guidance - Robot follows a path generated by an external device.
- EGM Path Correction - Programmed path is modified by measurements from external device.

For this project EGM Position Guidance will be used. The purpose of the module is to let the robot react fast to input from an external sensor.

EGM is an advanced tool that gives the programmer low level access to the robot controller by bypassing functions such as path planning. This makes it possible to read and write positions directly to the motion system at high rates, up to 250Hz. Position references can either be sent as joint variables or poses. Necessary filtering and state handling is performed by EGM Position Guidance. Expected time from when a new position is given to when it starts to affect the robot's position is typically around 20ms.

EGM Position Guidance also has some limitations. Since path planning is circumvented linear movements cannot be expected. Neither can a specific speed or time used for a movement be ordered. The robot path is created directly from the external program input and faulty inputs may result in sudden and damaging movements. If the robot encounters a singularity the movement will be stopped and it must be moved out of the singularity before a new EGM movement can be started.

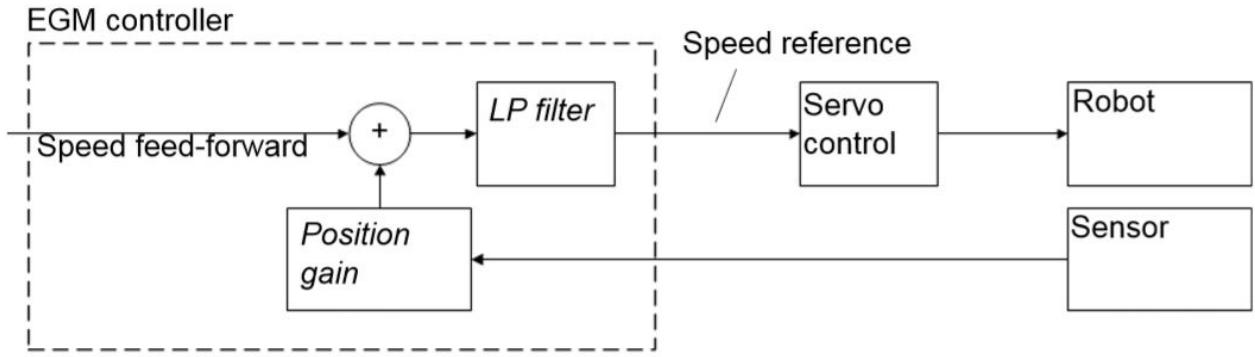


Figure 5: Simplified view of EGM control system. [22]

### Sensor protocol

The position reference source can be from a signal interface or through a sensor protocol. To utilize the high rate of EGM a fast protocol is needed. To this effect UDP is used as transport protocol along with Google Protocol Buffers [12] (Protobuf) for encoding. Protobuf is a way of serializing/de-serializing data very efficiently. The sensor acts as the server and communication must be started by the robot. After the first message data can be sent in both directions independently of each other. Figure 6 shows the flow of data through EGM module during operation.

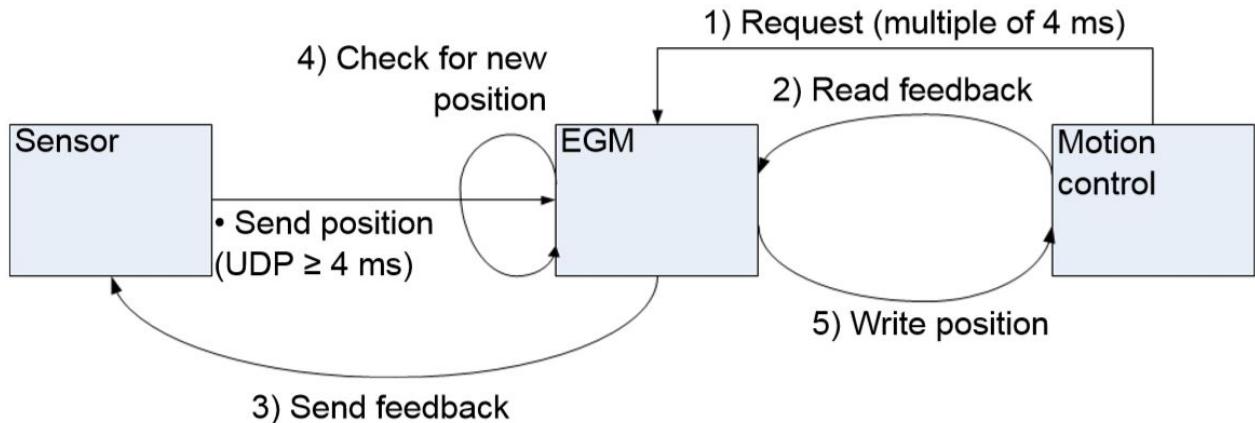


Figure 6: Data flow using UdpUc interface with EGM. [22]

### 2.2.4 Simplified Robot Programming - SRP

Simplified Robot Programming is a plugin to Robview. It is designed to let the user create paint programs for industrial paint applications fast and efficiently. The system consists of three main parts seen in figure 7.

- The Polhemus Liberty system handles the motion tracking through the use of electromagnetic fields.

- The teaching handle is the tool the operator uses to control the system and functions as a dummy paint gun.
- SRP plugin in Robview running on a PC.

The Polhemus Liberty Source generates an electromagnetic dipole field. This field is sensed by a sensor inside the SRP Teach Handle to find the position and orientation of the handle. To create paint programs the operator "paints" the object by moving the teach handle and holds the trigger button when paint should be applied. Recording can be started and stopped by buttons on the teach handle. The traced path and trigger points are then used by the SRP software to create the paint program. SRP also has built in algorithms for reducing redundant points in the path. For a straight line only the starting- and ending point is needed, which makes the path easier to edit afterwards. When the paint program has been created it is fully editable in other ABB tools such as ShopFloorEditor or RobotStudio.

The system works well and can cut programming time for creating new paint programs drastically. Using the system takes little training and will feel like second nature to an experience painter. The biggest challenge with the system is that the motion tracking uses electromagnetic fields. These can be affected by large metal objects in the working area and special precautions must be made when painting metal objects.

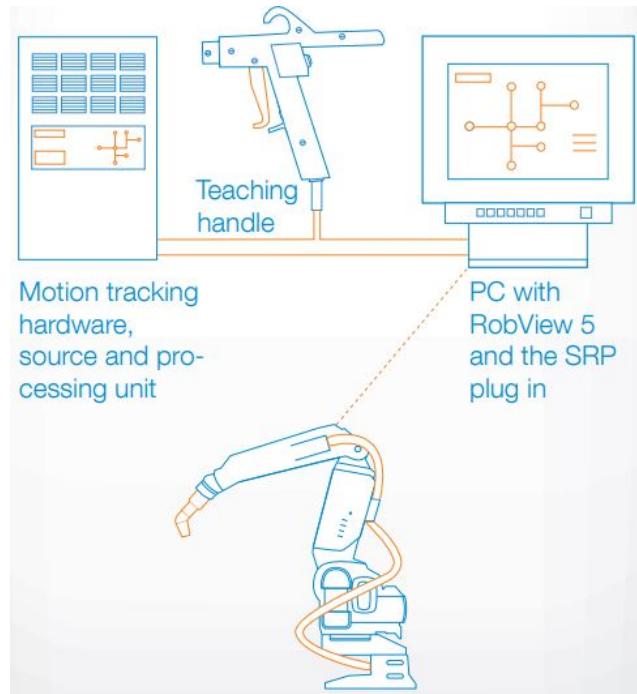


Figure 7: SRP system overview. [32]

end point is needed, which makes the path easier to edit afterwards. When the paint program has been created it is fully editable in other ABB tools such as ShopFloorEditor or RobotStudio.

The system works well and can cut programming time for creating new paint programs drastically. Using the system takes little training and will feel like second nature to an experience painter. The biggest challenge with the system is that the motion tracking uses electromagnetic fields. These can be affected by large metal objects in the working area and special precautions must be made when painting metal objects.

## 2.3 Programming tools

The programming language used for creating PC applications is C#. This was mandatory since SRP is written using this language. The IDE used is Visual Studio. It is a powerful programming environment with many helpful tools and robust debugger. Creating Graphical User Interfaces (GUI) is also simple with

the built-in graphics designer. As C# is ABB's preferred programming language for PC applications it also gives access to powerful tools for interacting with ABB robots.

The PC SDK library provides a framework for interacting with an ABB robot from a PC application. Communication with the controller is made possible by creating a controller object. The object is then used to log onto the controller.

The ABB.Robotics.PC namespace is divided into domains for the different aspects of the robot. The ones used in this project are described below.

- Rapid domain lets the application start/stop rapid execution and interact with variables in the RAPID modules on the robot.
- I/O System domain enables the application to interact with the inputs and outputs of the robot. For example outputs from the robot can be read by the application and be acted upon, or the program can set simulated input signals to control RAPID program flow.
- Configuration domain gives the application access to change the controller configurations.
- Motion domain lets you access the mechanical units of the system for example to read position data.
- File system domain gives access to the controller's file system. New program modules can be put in the controllers file structure and loaded into the current RAPID program to be run.

RobotStudio SDK is another C# library from ABB. It includes the ABB.Robotics.Math namespace that has data structures and functions for important mathematical structures when working with robots. Examples are 3/4D vectors, 3x3/4x4 matrices and quaternions. These structures have the necessary operators to interact with themselves and to convert between different structures.

# 3 Implementation

This section covers the implementation of the created systems. It is divided into one section for hardware and one for software. The project has mainly consisted of software development, and the hardware section mainly presents the choices of hardware and the reasoning behind them.

## 3.1 Hardware

For the choice of sensor a set of wanted criteria was made. The chosen system should if possible fulfill all of the criteria below.

- Good precision, preferably 1,5mm or better.
- High update rate. As low as 4ms if possible as this is the shortest possible time between position updates for EGM.
- Not affected by metals in the working area.

The sensor chosen for the project is the VR system HTC Vive described in section 2.1.1. This has been used by ABB in several other projects and has shown very promising results. It has been thoroughly tested to be used for external tracking of robots in a masters thesis by Kristian Sletten. [5] These results show that it has sufficient precision and reliability for the intended use. More testing to ascertain the viability of the system in this project will not be performed. Because of the familiarity to this system and its capabilities, other sensors were not considered.

To operate the systems a hand held controller (Human Machine Interface - HMI) is needed. This lets the user interact with the program and teach the robot. For this project three different options were considered for the HMI.

- Using the HTC Vive controller included in the VR set
- Using the proprietary Teach Pendant already used in SRP and mounting a Vive Tracker to handle tracking
- Creating a new tool with an integrated mount for the Vive Tracker

After consideration the first option was chosen. The HTC Vive controller is comfortable to hold and has many buttons that are readily available. It also has a haptic feedback unit which can give feedback to the user without relying on the screen. It is also the option that is least time consuming and leaves more

time to focus on software development. For a more in depth look at the pros and cons of the three options see discussion in section [5.1.2](#).

For development and testing an ABB IRB52 robot with a paint gun tool was made available. The systems should be compatible with any paint robot, so the choice of robot is not of great import. Virtual controllers and robots in RobotStudio have also been used for testing.

## 3.2 Software

This section covers the software implementation of the systems and is the main part of the project. The implementation of OpenVR which is shared between the two systems is presented first.

## 3.3 ViveTracker

Since the solution is used by both the systems, it is kept as simple as possible. It merely acts as an interface to the hardware. More advanced functionality such as storing paths and timing is left to the systems. As only the Vive controller is used in this project, only functions for the controller is implemented.

The solution consists of two classes. The **ViveData** class holds all the information. It has fields for the pose matrix, a vector and quaternion representation of position and the device's activity- and button states. **ViveTracker** is the class that connects to the hardware and handles communication. It has a ViveData field to hold the information and functions to interact with the hardware. The object can only handle tracking of one controller, but it is possible to use two controllers by creating two ViveTracker objects with different device indexes.

```
VRControllerState_t state = new VRControllerState_t();
TrackedDevicePose_t pose = new TrackedDevicePose_t();
_data._activityState = OpenVR.System.GetTrackedDeviceActivityLevel(_deviceIndex);
bool result = OpenVR.System.GetControllerStateWithPose(_trackingUniverse, _deviceIndex,
    ref state, System.Convert.ToInt16(System.Runtime.InteropServices.Marshal.SizeOf(state)), ref pose);
```

Listing 1: Code excerpt from ViveTracker.UpdateData().

The *Initialize()* function opens up a connection to the hardware by calling *OpenVr.Init(\*)*. To get new information from the controller *UpdateData()* is used. The controller's state is retrieved by the code in listing 1. Then some conversions are performed to be able to use the information. The pose of the controller is given as a 3x4 pose matrix. This is converted to a 4x4 Matrix in the ABB.Robotics.Math namespace. This namespace also has functions to extract

the 3D vector and quaternion representation of the position which is what is used by ABB robots. The controller's reference frame is centered around a point in the middle of the controller approximately where the menu button is located. The ViveTracker object also has an optional 4x4 transformation matrix that can be used to move or rotate this reference point for example to the tip of the controller.

The button states are stored in the VRControllerState\_t structure. It has a 64 bit unsigned integer with one bit for each button that is pressed. To check whether a button is pressed a logical & operation with the correct button mask is used. The buttons used and their names are shown in figure 8. To get access to more buttons the trackpad was divided into four. The trackpad measures the finger's position in 2D. By checking if the x or y axis respectively is larger than a threshold the position of the finger is determined. This is combined with the button mask for the push button to get four new buttons called dPad- up/down and left-/right. This gives a total of seven buttons that can be used when programming.

```
public void HapticPulse(uint duration)
{
    Task.Run(() => {
        for (uint i = 0; i < (duration / 5); i++)
        {
            HapticSinglePulse();
            Thread.Sleep(5);
        }
    });
}
```

Listing 2: Program excerpt: *HapticPulse(\*)*

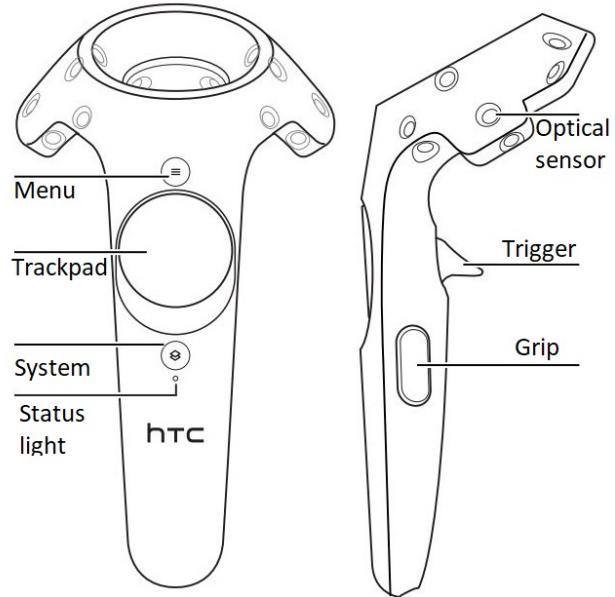


Figure 8: Vive controller with button names. [16]

*HapticPulse(\*)* makes the controller vibrate to give feedback to the user. The haptic pulse in OpenVr has a max duration of 3900  $\mu s$  and can only be triggered once every 5ms. To get longer durations a new task that runs independently of the main task is started and triggers a haptic pulse every 5ms for the desired duration. See code in listing 2

## 3.4 Simplified Robot Programming

New functionality implemented in SRP is based on wishes expressed by ABB:

- Integrate the new sensor HTC Vive into the system.
- Adding a new recording mode called point to point.
- Automatically check if the recorded path can actually be run on the robot.

The implementation of these new features will be described in this section.

The SRP plugin is too comprehensive to be described in full in this report. Only the changes made to the program in this project will be presented here. All changes have been made with the thought in mind that it should be backwards compatible with the existing sensor.

### 3.4.1 System overview

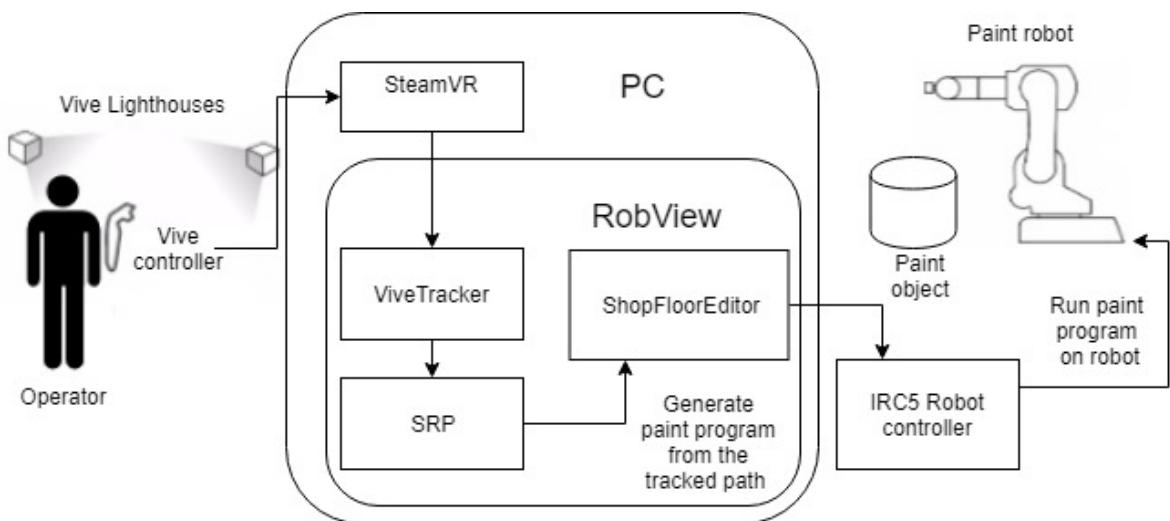


Figure 9: Block schematic showing the different elements.

Figure 9 shows an overview of the system. The operator controls operation through the hand held Vive controller. Pose of the controller and button presses are retrieved by ViveTracker through the SteamVR software. A path is recorded by the operator tracing the object with the controller, holding the trigger button when paint should be administered. The recorded path is then used to create a paint program in RAPID code which can be viewed in for example ShopFloorEditor or RobotStudio. This paint program can then be transferred to the robot and used to paint the object. A screenshot of SRP's GUI can be seen in figure 10.

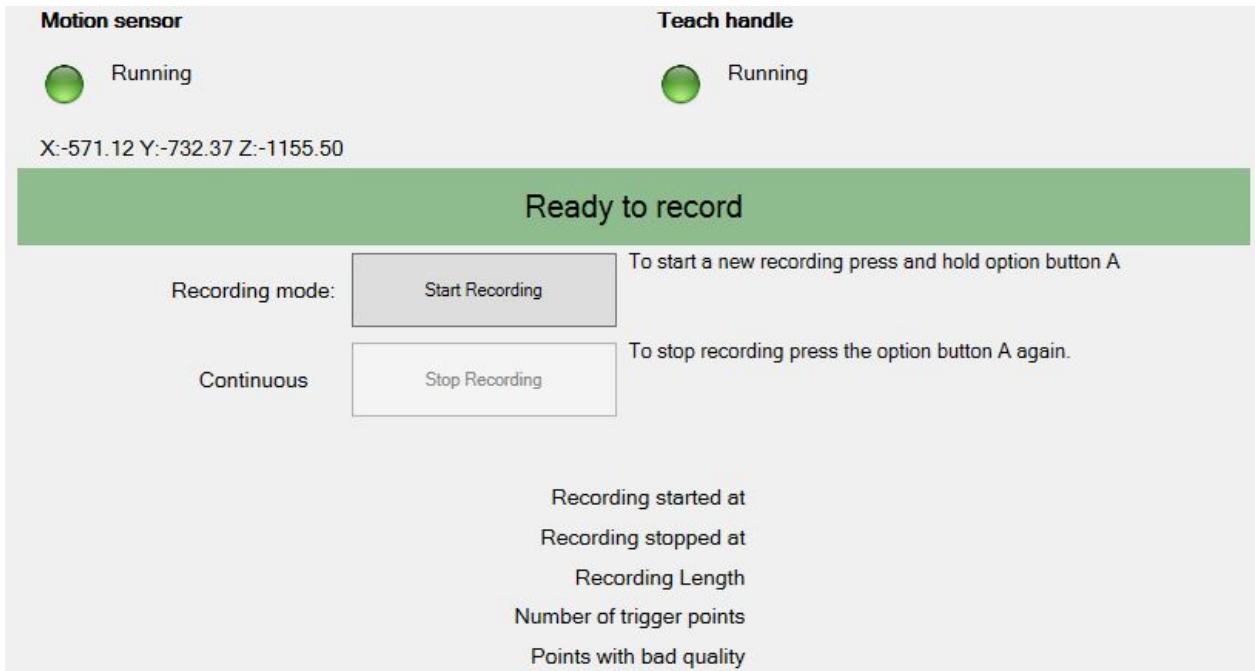


Figure 10: View of the SRP GUI ready to start new recording

### 3.4.2 Implementing new sensor

SRP has been programmed with the possibility of adding new sensors. It uses an interface **IMotionSensor** to facilitate this. To use the Vive as the sensor a new class **MotionSensorHtc** which implements this interface is created. The user can then choose which sensor to use in the program's settings.

When the MotionSensorHtc object is created by the program, a new Vive-Tracker object is made and instantiated and initialized. It then starts an update loop on a separate thread that runs *UpdateData()* every 4ms. Every time the updated position changes an event is created with the new sensor information. Other classes in the program subscribe to this event to react to changes in position.

Three buttons are used to control the program from the controller:

- The trigger button on the Vive controller is equivalent to the Teach Handle's trigger. It lets the user trigger paint on and off while recording a path.
- The menu button replaces the A button on the Teach Handle. This starts and stops recording.
- The grip button is the same as the Teach Handle's B button. It adds an extra option to trigger events other than paint on and off along the programmed path.

The new motion sensor object also includes an implementation of the Vive controller's vibrate function. Vibration is triggered when recording starts and stops so the user does not need to see the screen to confirm the recording state.

The existing solution currently has one recording mode, continuous. In this mode the operator records a continuous path by tracing the path with the teach handle. The point to point recording mode instead lets the operator record discrete points and the program creates a linear path between them. For simple objects where only straight lines are needed this can be more efficient. This can also lead to higher precision because it negates unwanted shaking of the hand while recording.

### 3.4.3 New recording mode

To implement point to point recording an option to change recording mode has been added to the settings as seen in figure 11. A label showing the current operating mode has also been added to the SRP main window as seen to the left of the "Start recording" button in figure 7. Continuous mode uses a class called **ContinuousPathProcess** to handle the recording process. A new class **PointTo-**

**PointProcess** is created to handle the recording process for point to point mode. This class has functions to create new recording, start/stop recording and to add points to the recording. While in the continuous case every new position is added to the path, for the point to point mode a new point is added when the user presses the A/menu button. Whether the trigger is pressed or not when a point is recorded determines if paint should be triggered on or off at this position. When recording mode is changed, the current process object is disposed off and a process object for the new recording mode is instantiated. Where the displayed messages of the UI differ between the two modes, a check for recording mode has been added and the correct text is displayed.

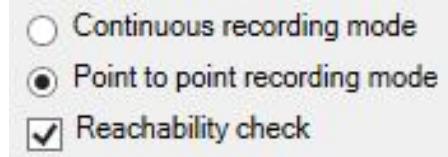


Figure 11: New settings options in SRP

### 3.4.4 Reachability checking

Being able to check if a path is invalid without having to run it on the robot or check it in RobotStudio can save a lot of time when creating new paint programs. This will not give any information about the quality of the paint job, but can confirm that it is possible or safe to run on the robot. To facilitate this, a framework for checking if added points are reachable has been created.

This framework includes a **IReachabilityCheck** interface which implements the function *CheckPointReachable(\*)*. This function takes a point as input and returns a boolean value whether the point is reachable or not. Implementing this interface lets any new reachability checker easily be integrated in the system.

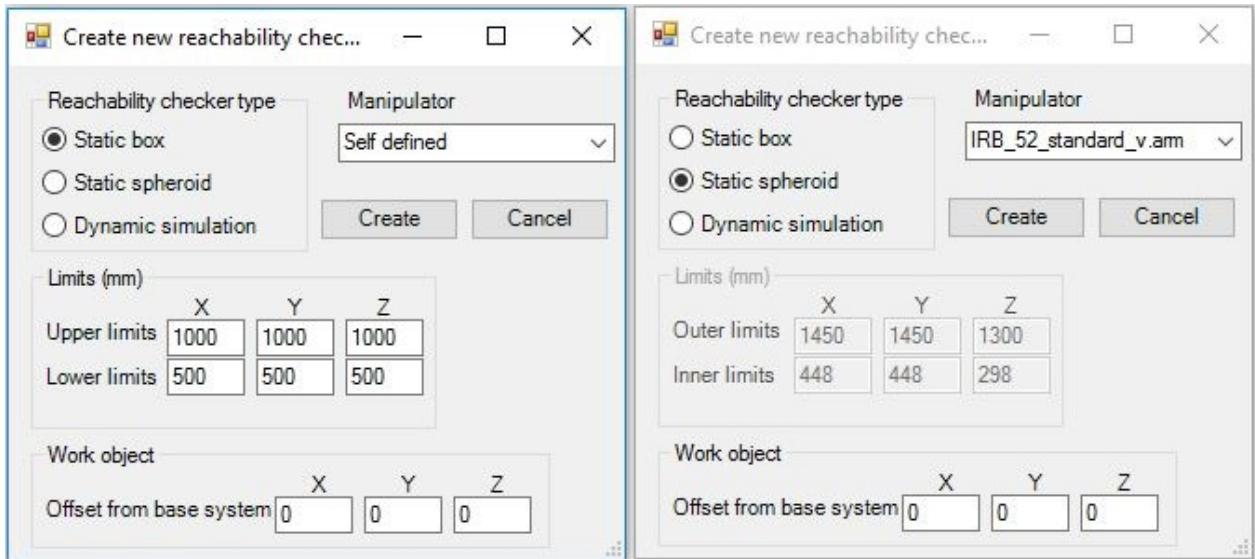


Figure 12: Window for creating new reachability checker.

Figure 11 shows the option to turn reachability check on. When this is selected the window shown in figure 12 will appear when the settings window is closed. The form lets you choose which type of reachability checker to be employed and set necessary parameters. The parameters can either be self defined by the user or a set of premade limits for some common manipulator arms can be made. The limits are defined from the robot's base coordinate system, but the reachability check in the program is based on the position in the current work object. The offset of this work object in relation to the robots base must therefore be specified for the reachability check to be valid. The different reachability checker types are described in the following.

**Static box** checker creates a right prism with the upper and lower limits given as parameters. To be reachable the point must be within the box. The *CheckPointReachable(\*)* function compares the input position's X,Y and Z-components to the upper and lower limits and returns true if all checks are successful.

**Static ellipsoid** creates an inner and an outer ellipsoid based on the given parameters. To be reachable the point must be outside the inner ellipsoid and inside the outer ellipsoid. The general formula of an ellipsoid is given by: [2]

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$$

where  $a, b$  and  $c$  are the semi-axes of the ellipsoid. To check whether a point is inside or outside the ellipsoid the position's X, Y and Z-coordinates are plugged into this formula. If the result is larger than 1 for the inner ellipsoid and smaller than 1 for the outer ellipsoid the point is reachable. An example with inner and outer limits has been made for the IRB52 robot based on the drawings in figure 13.

An option for a dynamic model of the robot to check reachability was also added to the form. Implementing such a dynamic model was outside the scope of this project.

Figure 19 Working area IRB 52, long vertical arm

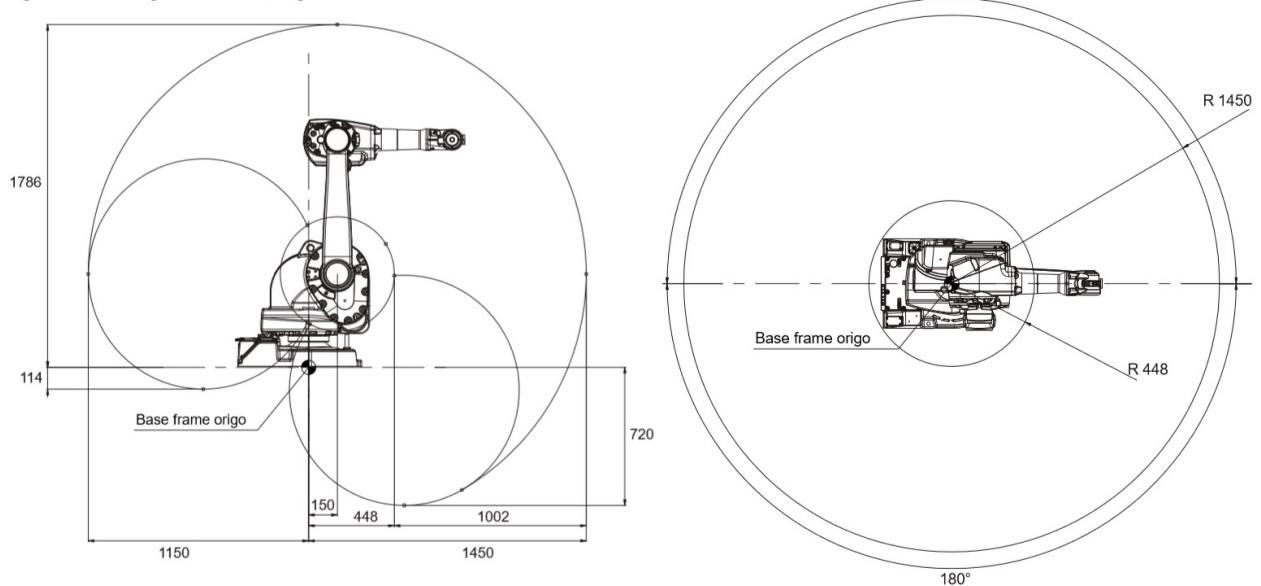


Figure 13: Drawing showing the working area of an IRB52 robot [28]

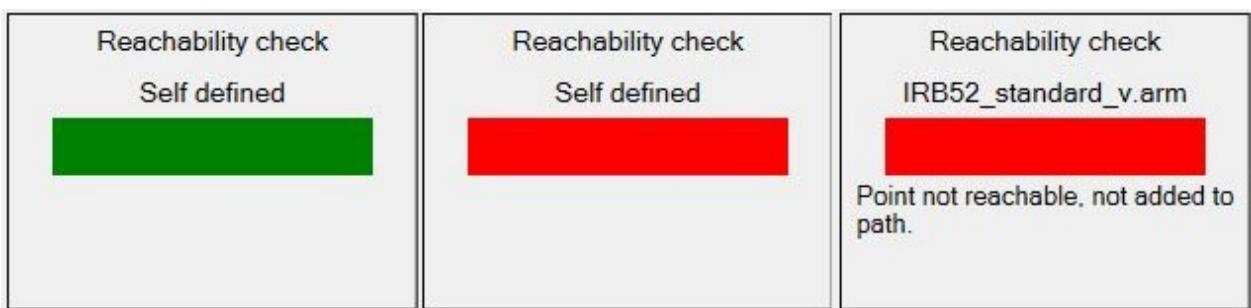


Figure 14: Reachability checker showing different states.

When the reachability check has been activated and the checker created, the window seen in figure 14 will be visible in the SRP window. It shows whether the point is reachable or not and the controller also starts vibrating when it is outside the reachable area. If the controller moves outside the reachable area during a continuous recording, the controller will vibrate to give feedback to the operator of an error and recording will be stopped. For point to point mode it will not add a point to the path if it is not reachable and the message in the rightmost window of figure 14 will be displayed.

## 3.5 Online Teaching

The goal of the second task is to have the robot paint the object in real-time while the path is being recorded. This gives immediate feedback about the paint program's quality and whether the robot can reach all the points along the path. Some wishes for the system's capabilities are listed below:

- Operator controls robot's movement in real-time.
- Generated paint program is based on feedback from the robot.
- Continuous and point to point recording mode.
- Usable with a wide range of robots.

The system Online Teaching is created to fulfill this task. It consists of a PC application and a RAPID program on the robot. Program logic is mostly contained in the PC program, and the robot side mainly handles robot movement.

### 3.5.1 System overview

An overview of the full system can be seen in figure 15. As for SRP the program is controlled through the hand held Vive controller. ViveTracker retrieves the controller's pose and state through SteamVR. Position data is handled by EGMSSensor which sends position reference to the robot controller and receives feedback data at a rate of 250Hz. Button presses are handled by the GUI which in turn controls RAPID program flow through changing RAPID variables directly or setting simulated digital input signals to the controller. When robot movement is activated by the operator, the robot will mirror the controller's movement in both position and orientation.

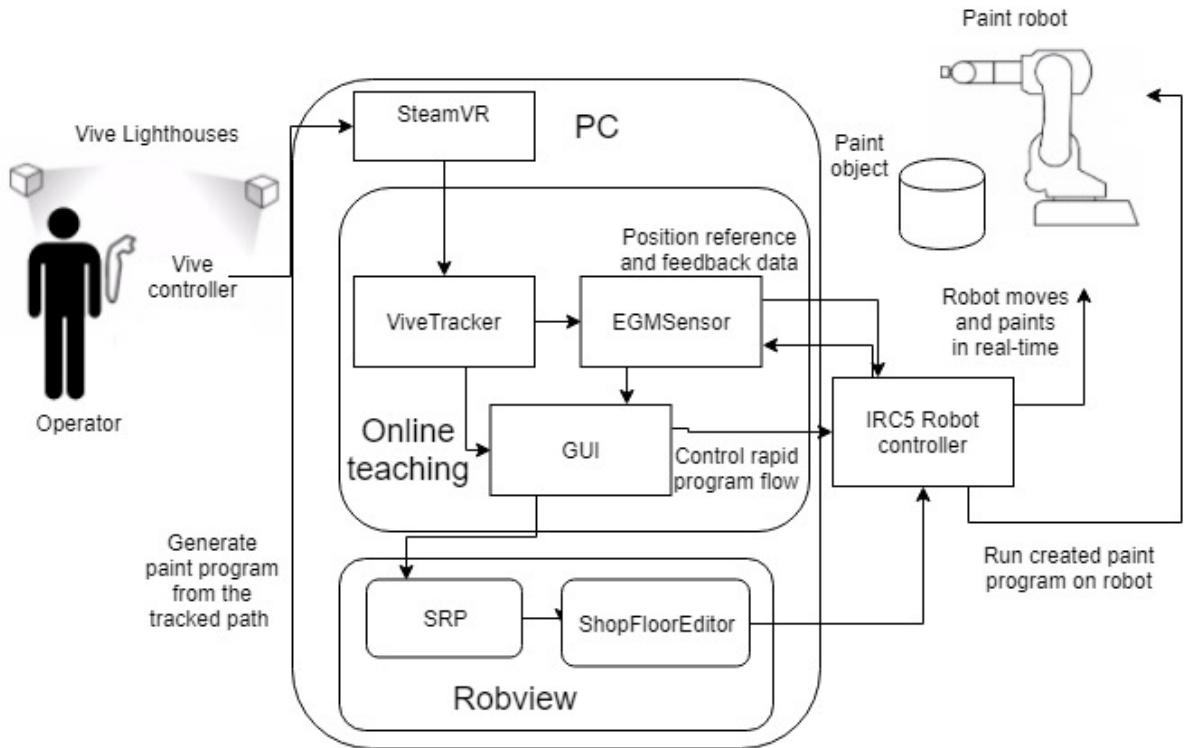


Figure 15: Block schematic showing the different elements.

To make a recording the feedback data from the robot is stored in the PC application along with brush and time information. The recording is saved as a text file which can be read by the SRP plugin of RobView and create a full paint program in RAPID code. This can then be loaded onto the robot and used to paint objects.

### 3.5.2 Robot program

The robot part of implementation consists of setting up Externally Guided Motion and writing the RAPID program to run on the robot. How to set up and use EGM is found in [22] from page 344 and onwards. This also includes sample RAPID code that is used as a basis. To use EGM the robot controller needs to be properly configured. This can be done by connecting to the robot controller through RobotStudio and accessing the controller's configurations. The first is to setup the UDP communication as seen in figure 16.

Figure 16: Configuration of UDP communication.

"Remote Address" is the IP address of the PC running the application, and the chosen port must be the same both on the robot and PC side.

Name	EGMsensor
Type	UDPUC
Remote Address	192.168.0.181
Remote port number	6510

Figure 17 shows how motion data for EGM is configured. It defines what should happen when the movement is stopped and some important tuning parameters. To facilitate communication between robot and PC application the signals seen in figure 18 are also created. These are simulated digital input signals which can be set or reset through PC SDK's I/O-System domain. Connecting the signals to interrupts in the RAPID code lets the PC application control program flow on the robot.

Name	EGMdata
Level	Filtering
Do Not Restart after Motors Off	<input type="radio"/> Yes <input checked="" type="radio"/> No
Return to Program Position when Stopped	<input type="radio"/> Yes <input checked="" type="radio"/> No
Default Ramp Time	0,1
Default Proportional Position Gain	20
Default Low Pass Filter Bandwidth	6

Figure 17: Configuration of EGM motion data.

Name	Type	Value	Min Value	Max Value	Simul:	Network	Device	Device Mapping	Category
TriggerButton	DI	0	0	1	Yes	<none>	<none>		Online teaching
GripButton	DI	0	0	1	Yes	<none>	<none>		Online teaching
DpadButton	DI	0	0	1	Yes	<none>	<none>		Online teaching
MenuButton	DI	0	0	1	Yes	<none>	<none>		Online teaching

Figure 18: Simulated digital input signals.

## Tuning of EGM

The most important tuning parameters of EGM are described below. Values for this project have been chosen based on tests performed in [6].

- **Default Ramp Time** defines the time from when EGM movement is started until it should be correctly following the external position reference. Short ramp time can cause increased strain on the robot when movement is started.
- **Proportional Position Gain** adjusts the weight of the external sensor position data when determining speed reference sent to robot (see figure 5). Higher values gives faster response, and the max value of 20 is used here.
- **LPfilter** describes how heavily the position reference should be filtered before being applied. Lower values give faster reaction to changes but more oscillatory motion. A value of 6 is used in this project.
- **MaxSpeedDeviation** defines the maximum combined speed of all joints in degrees/second. This parameter is set in the EGMActPose command described below and a value of 50 has shown to give good results.

## RAPID

The RAPID program handles communication with the PC application and the robot's movement. It consists of variables, procedures (similar to functions or methods in other languages) and trap routines that are started by interrupts (similar to event handlers). Communication with the PC application happens in three ways.

- Position data is sent at a high rate through the EGM sensor.
- RAPID variables can be changed directly from the PC application by using PC SDK's Rapid Domain.
- Button presses on the Vive controller sets or resets simulated digital input signals to the controller. These input signals trigger interrupts which controls program flow of the RAPID program.

The RAPID program consists of four modules. **EGM** is the largest and contains everything that is shared between the different modes. It has work object and tool definitions, defines all variables for communication with the PC application and defines interrupt identities. When RAPID execution is started the robot moves to a given start position, sets up UDP communication and EGM then switches to the module corresponding to the mode chosen in the PC program. The procedures of the EGM module are described below:

- *SetUpUDP()* starts the UDP communication with the PC application through the EGMSsetupUC command. The EGMSensor in the configurations defines which IP and port to use.
- *EGMSetup()* initializes EGM by defining the work object, tool and other parameters with the EGMActPose command.
- *EGMRun()* starts the actual movement of the robot by using the EGMRunPose command. See code in listing 3.
- *PlayRecordingProc()* lets the operator preview the recorded path by running it on the robot.

The three other modules - **Testing**, **Continuous** and **PointToPoint** - each corresponds to a mode in the PC application. They are similar in structure and the Continuous module will be used as an example.

The main procedure is started from the EGM module. It starts by setting up the necessary interrupts and connecting them to trap routines. Then it

```

! Runs the EGM movement until convergence condition
! has been met or EGM movement is stopped
PROC EGMRUN()
    EGMRUNPOSE egmID1,EGM_STOP_HOLD
    \X\Y\Z
    \RX\RY\RZ
    \CondTime:=10
    \RampInTime:=0.05;
ENDPROC

```

Listing 3: RAPID code excerpt showing EGMRUNPOSE command.

runs a while loop waiting for input from the PC application to either start an EGM movement or to run a recorded path. This module has three trap routines that are activated from the controller. Pressing/releasing the trigger button activates/deactivates spraying of paint. The grip button's trap routine gives the operator "control" of the robot by starting an EGM movement. A subsequent press of the button will stop the movement.

### 3.5.3 PC application

#### EGMsensor

The setup of the EGM sensor protocol follows the guide on page 360 of [22]. A .proto file is provided by ABB. This file is then compiled into C# code using protobuf-csharp-port [13]. The generated C# file egm.cs contains the necessary definitions and functions for serialization and de-serialization of the data packets to be sent. It defines the data structures EgmRobot and EgmSensor which are sent from the robot and sensor respectively. The header is common for both structures and includes the sequence number, timestamp in milliseconds and whether it is sent from the robot or sensor. A packet from the robot carries the position feedback from the robot, while the sensor packet includes the planned position the robot should go to next.

An example file egm-sensor.cs is provided by ABB. This file is adjusted to fit the needs of this project. To use it a sensor object is created. When started it starts listening for communication on the given port, in this case 6510, the same port as in the robot's configurations. After communication has been established by the robot controller, data messages will be sent both ways independently of each other until communication is halted.

## Menu

The simple menu of Online Teaching is shown in figure 19. It gives access to the different parts of the application. To be able to use the application it must first be connected to the robot. The "Connect to robot" button opens the window shown in figure 20. This window loosely follows the "Create a simple PC SDK application" guide in [21]. It shows all the available robot controllers on the network and lets the user choose which to connect to. This instantiates a controller object which is used to log on and acts as the interface to the robot controller.

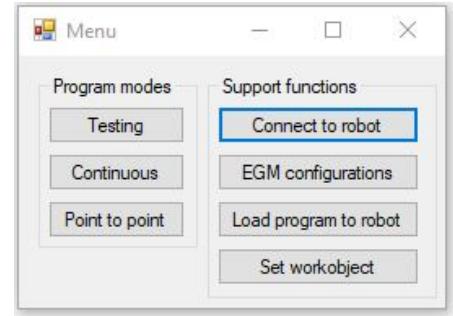


Figure 19: Screenshot showing menu of Online Teaching.

Network scanning window						
IP Address	ID	Availability	Virtual	System name	RW Version	Controller name
10.47.89.130	CBS	Available	False	RW6.07.01_t	6.7.0.1	CBS Lab
10.47.89.106	VanPainter	Available	False	RW6.07.1008	6.7.0.1	VanPainter
10.47.89.90	NinasRack	Available	False	RW6.07.1006	6.7.0.1	NinasRack
10.47.89.35	C536	Available	False	RW6.07.0095	6.7.0.0	ElevatedRail
10.47.89.122	IRB52_TiltedIps	Available	False	IRB52_TiltedIps	6.7.0.1	IRB52_TiltedIps
10.47.89.228		Available	False	2KP_Gun	6.7.0.0	
10.47.89.85	IRB52	Available	False	RW6.07.1008	6.7.0.1	IRB52
10.47.89.226	ABB_Test	Available	False	C523_RW5.15.9150	5.15.0.16	C523_IRB5400-13_PaintLab
10.47.89.188	C537	Available	False	RW6.07.0108	6.7.0.0	C537_IRB5500_CBSII_PAINT...
10.47.89.210		Available	False	RW6.07.1008	6.7.0.1	
10.47.89.76	NGAC1	Available	False	RW6.07.1008	6.7.0.1	C530
10.47.89.32	ABB	Available	False	RW6.07.01	6.7.0.1	GAP-2K-PaintLab
10.47.89.180	C509TestRack...	Available	False	RW5.15.3034	5.15.0.3	C509
10.47.86.249	1200-101067	Available	False	1200MotionTest	6.8.0.0	
10.47.86.250	DongXi	Available	False	Norway TesttrackRW...	6.8.0.0	
10.47.89.80	Small_Door_O...	Available	False	RW6.07.1008	6.7.0.1	Small_Door_Openner
10.47.89.117	C539 - RCC	Available	False	RW6.06.0115	6.6.0.0	C539 - 5400 RCC

Connect   Refresh   Cancel

Figure 20: Netscan showing available robot controllers on the network.

The two next support functions are used to prepare the robot controller the first time it is used with the PC application. As shown in the previous section some configurations must be made before EGM can be used. All these configurations are added automatically through the use of PC SDK's configuration domain by pressing the "EGM configurations" button. The IP address of the PC running the application is the only parameter that changes. It is found automatically by code suggested in [10]. The next button then transfers the program modules from the PC to the controller's file system and loads it into the current

task. The controller configurations and RAPID program is then ready to be run without needing to log onto the controller through RobotStudio.

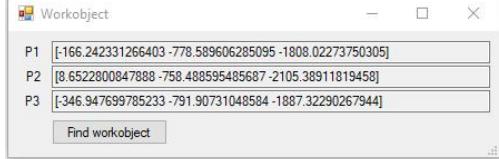


Figure 21: The work object creation form.

The "Set workobject" button opens the form shown in figure 21. This lets the user create a new coordinate reference frame for the HTC Vive that corresponds to the robot's. This ensures that the robot will follow the controller's movement correctly. The form asks the user to define three points by moving the controller and pressing the trigger button.

First an arbitrary initial point is recorded.

The second point should then be in the robot's positive x-axis and the third point in the robot's positive y-axis relative to the first point. The rotation matrix is then found from the code in listing 4. When the 3D position vector from the Vive controller is multiplied with the found 3x3 rotation matrix it will give the position in the robot's coordinate system.

```
Mainform.staticMatrix4 = Matrix4.FromThreePoint(P1, P2, P3);
Matrix3 rot = new Matrix3(Mainform.staticMatrix4);
Mainform.staticMatrix3 = rot.Inverse();
```

Listing 4: Code excerpt showing creation of transformation matrix.

The buttons on the left lets the user choose the preferred mode. Testing is used for development and testing and will not be explained in detail. It shows both the Vive and robot positions and other relevant debug information in the GUI. The other two will be presented in the following sections.

### 3.5.4 Continuous mode

The functionality of the program is handled by a windows form. It holds all the variables and functions. The controller object and work object from the menu is passed to the form when created. Initialization of the form creates a new EGMSSensor object which starts listening for a connection attempt from the robot. A ViveTracker object is created to start tracking of the controller. The controller object is also used to log onto the controller, start RAPID execution and get the digital signals needed to interact with the RAPID program. When initialization is complete it opens the Graphical User Interface.

Figure 22 shows the GUI of continuous mode with the different recording states. The sensor status reflects the activity state of the Vive controller while the robot status is whether the RAPID program is running on the robot. Both must be green in order to start recording. The brush selector sets the correct brush to be used by changing a variable in the RAPID code. If a recording has

been made there are buttons to save the recording, preview by running it on the robot or reset the recording. The GUI is updated every 100ms by a timer.

Program flow is handled by a timer with an interval of 4ms. As the smallest interval of the built in windows timers was up to 15 ms, a better timer was needed. A time from the custom made MicroLibrary [9] was used. This timer counts the elapsed  $\mu$ s between intervals and has much better resolution at a cost of more computational resources being used. The timer method runs *UpdateData()* to get new information from the controller then checks if any of the buttons have changed state since last update.

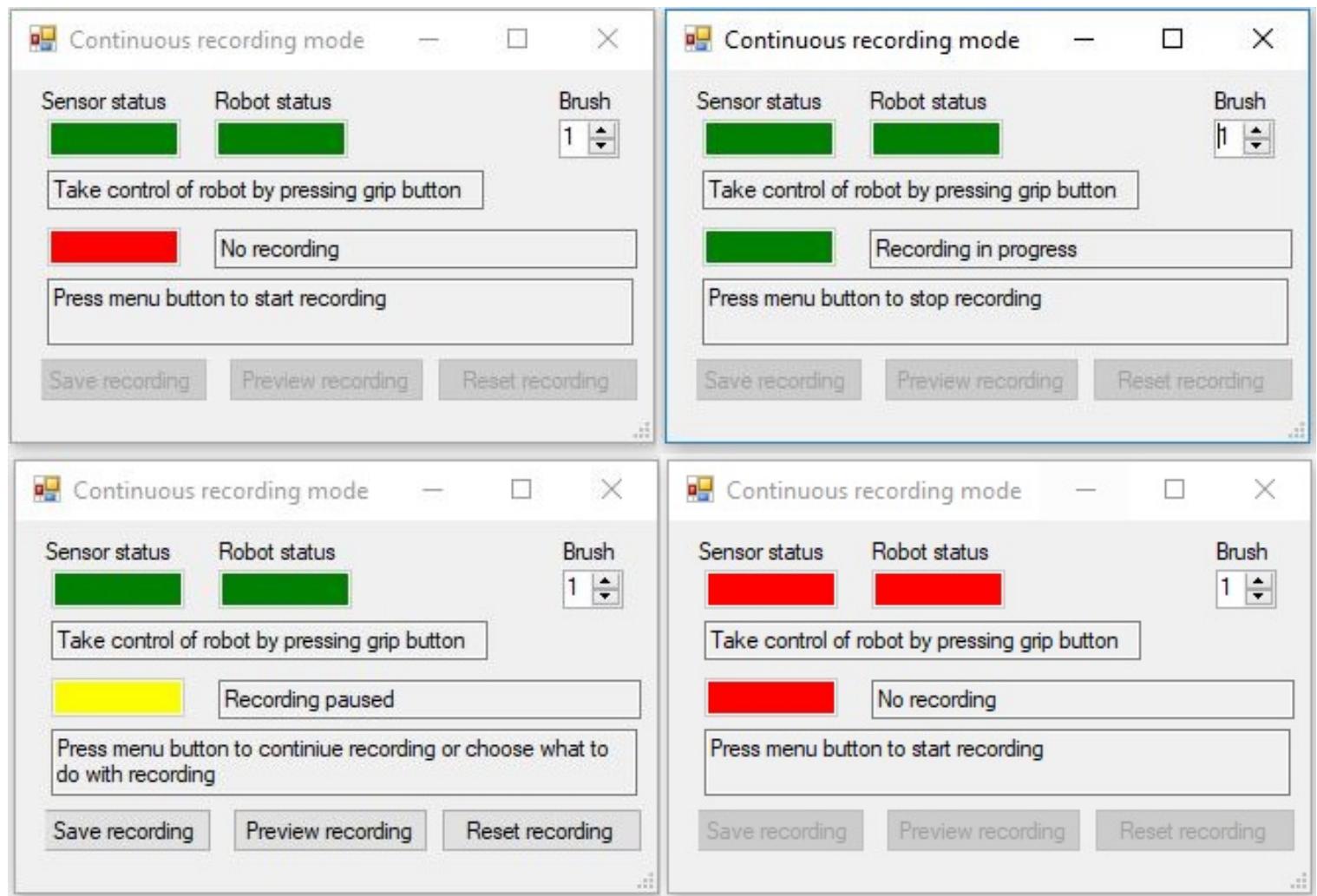


Figure 22: GUI screenshots showing different states of continuous recording mode.

## Calibration

To "take control" of the robot the operator presses the grip button on the controller. This makes the robot follow the movement of the controller in a 1:1 ratio. The robot's and controller's positions are not absolutely mapped in relation to each other. When the grip button is pressed the current position is used as a reference point and position data is defined relative to this point. The code in listing 5 shows how the calibration is computed. First the current position of the Vive controller is stored in the CalibVive variable. Then the position of the robot is obtained from the robot, converted to the right format and stored in OffsetRobot. These variables define the reference point for the position part of the pose. Along with the work object predefined by the operator, which is a 3x3 rotation matrix, these are used when calculating the position to be sent to the controller as seen in listing 6.

```
// Position calibration
CalibVive = tracker.Data.Position;
currentPos = controller.MotionSystem.ActiveMechanicalUnit.GetPosition
    (ABB.Robotics.Controllers.MotionDomain.CoordinateSystemType.WorkObject);
OffsetRobot.x = currentPos.Trans.X;
OffsetRobot.y = currentPos.Trans.Y;
OffsetRobot.z = currentPos.Trans.Z;

// Orientation calibration
Quaternion Robot = new Quaternion(currentPos.Rot.Q1, currentPos.Rot.Q2, currentPos.Rot.Q3, currentPos.Rot.Q4);
Quaternion Vive = tracker.Data.Orientation.Inverse();
transformViveRobot = Vive * Robot;
```

Listing 5: Code excerpt showing position and orientation calibration.

Calibration of orientation is done separately. The current orientation of both the Vive controller and the robot is found as quaternions. This is used to find the transformation between the two orientations using formulas described in section 2.2.1. The resulting quaternion is used when calculating new robot orientations as seen in listing 6. The next time the sensor sends position data to the robot controller PlannedPos and PlannedOrient will be used.

```
// Position data sent to robot
sensor.PlannedPos = wobj * ((tracker.Data.Position - CalibVive) + OffsetRobot);
sensor.PlannedOrient = tracker.Data.Orientation * transformViveRobot;
```

Listing 6: Code excerpt showing data sent.

## Recording

When EGM is running and the robot is following the controller's movement, recording can be started by pressing the menu button. The controller vibrates to confirm that recording has started, a timer is started to keep track of the time and a boolean to start recording is set to true. While an EGM movement

is active the robot sends position feedback to the PC application every 4ms. When recording this feedback is stored in a list of RobTargets along with the current time and the brush number. When the button is pressed again recording is stopped and the buttons to preview/save recording are enabled.

The recorded path can be previewed on the robot without creating a full RAPID program. This is done by writing the position and brush value to arrays stored on the robot. The path is then run by using *PlayRecordingProc()* procedure in the EGM module. To create a paint program from the recorded path, the SRP plugin in RobView is used. The recording is saved in a text file in a comma separated value (CSV) format. This text file can be loaded in the SRP plugin and the RAPID code for the paint program can be generated automatically.

The sequence diagram in figure 23 shows how the operator, PC application and robot controller interact during a simple recording.

### 3.5.5 Point to point mode

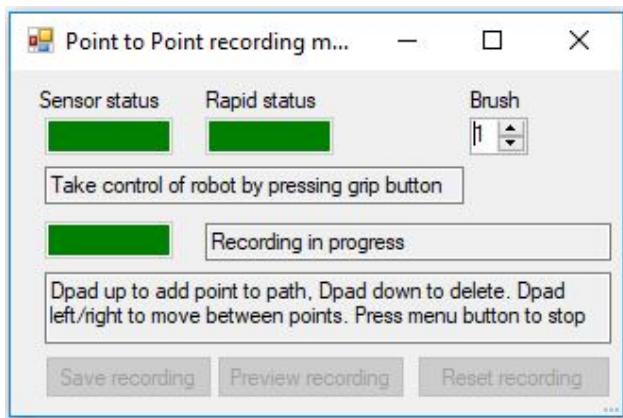


Figure 24: GUI screenshot showing point to point mode during recording.

The point to point mode uses the same structure as the continuous mode. It uses a timer to control program flow and calibration is done in the same way. The difference is in regards to recording as can be seen in the recording help text in figure 24. After recording is initiated by pressing the menu button, the operator uses the trackpad's four individual buttons to record points. An index is used to keep track of current position in the recording.

Pressing up on the trackpad adds

the current position of the robot to the path at the current index along with the trigger status. The position is retrieved directly from the robot in the same way as when doing the calibration in listing 5.

To delete a point the operator presses down on the trackpad. The point at the current index is removed and index is reduced by one. To move the robot to the correct position a variable in RAPID is updated with the correct new position. Then the signal DpadButton is set to trigger the interrupt to run the correct trap routine seen in listing 7. First the EGM movement is stopped, then the robot is moved to the correct position with the MoveL command. As

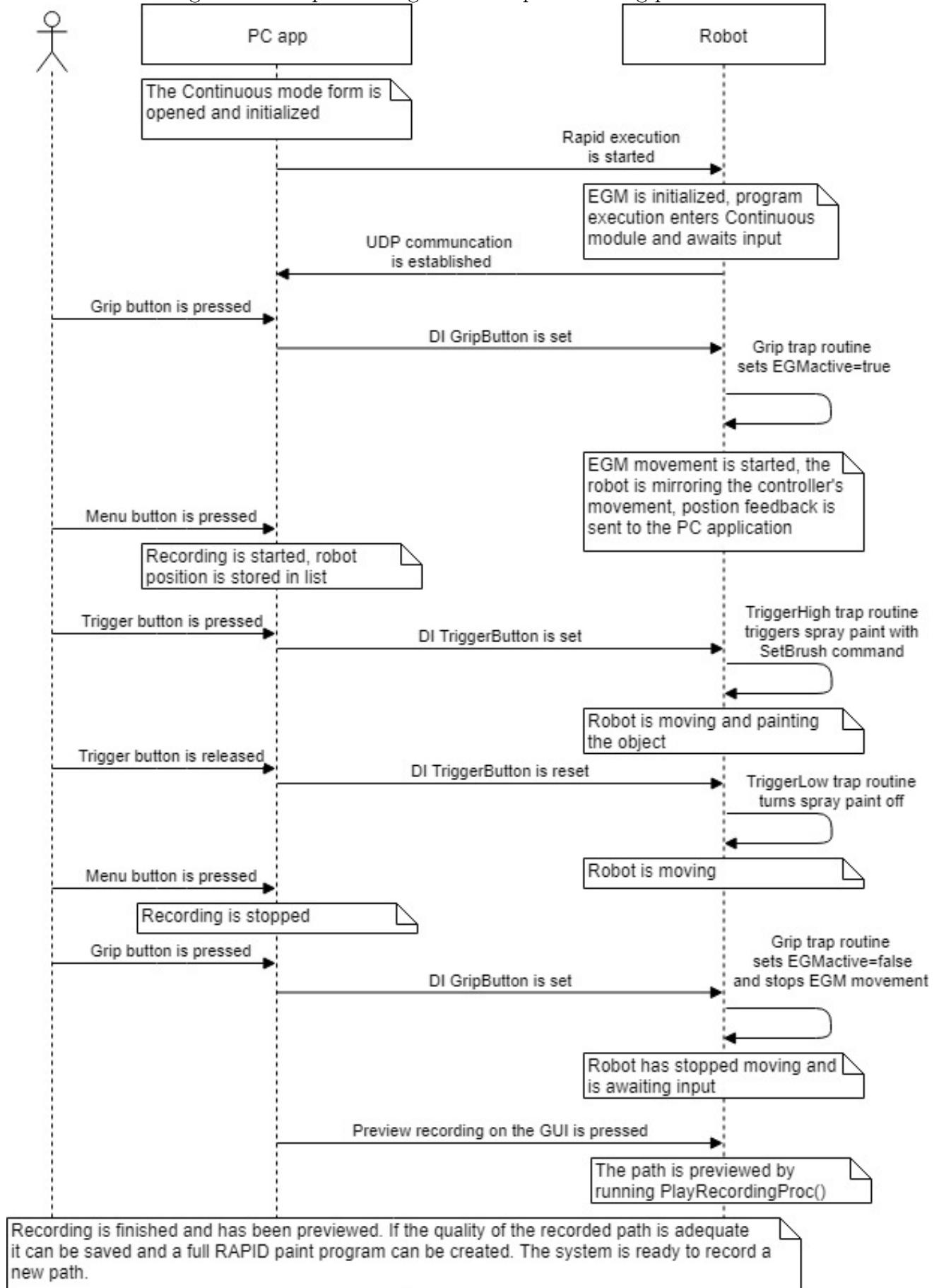
EGM movement is stopped the operator must press the grip button to again take control of the robot.

```
! Stops EGM movement and moves to the point in the recorded path
TRAP P2PDpad
    EGMStop egmID1, EGM_STOP_HOLD;
    EGMactive := FALSE;
    MoveL P2Ppos,baseSpeed,baseZone,EGMtool,\WObj:=EGMwobj;
endtrap
```

Listing 7: Code excerpt from PointToPoint module showing Dpad trap routine.

Left and right buttons on the trackpad lets the operator move between recorded points. This is done by incrementing or decrementing the index by one, updating the position in the RAPID program and using the Dpad trap routine to move the robot to the new position.

Figure 23: Sequence diagram of simple recording process.



## 4 Tests and results

To verify proper operation of the systems some simple tests are performed. The tests seek to test the functionality of the systems and are mainly qualitative in nature. Tests to specifically test the hardware were not designed. However, all the tests rely on the hardware and will discover if there are challenges with the chosen solutions.

### 4.1 Simplified Robot Programming

The general functionality of the program is tested first. A Vive controller and Lighthouse is connected through SteamVR and RobView with the SRP plugin is running. The controller is recognized by the program and its position is being tracked. Pressing the menu button will start/stop recording and the trigger button is in charge of paint triggering. It is possible to change between the two recording modes and the UI is updated to reflect the correct mode.

The main part of the test consists of making some simple paint programs. Two paths are recorded for each mode and then displayed using ShopFloorEditor. The paint programs are also loaded onto a virtual IRB52 robot to check that it will run on the robot.

### 4.1.1 Continuous recording

The first recorded path is to simulate painting a flat surface. The controller is held against a level surface and the orientation is kept the same for the whole recording. Moving the controller and holding the trigger button when paint should be sprayed the path is recorded.

For the second path an imaginary 3D object is painted. The controller is moved along a level surface before being tilted 90 degrees and moved straight up. It is then moved down along a semi-circle until it reaches the level surface. The controller is then moved to perform the same motion perpendicular to the first path. The results from the two recorded paths are seen in figure 25 and 26.

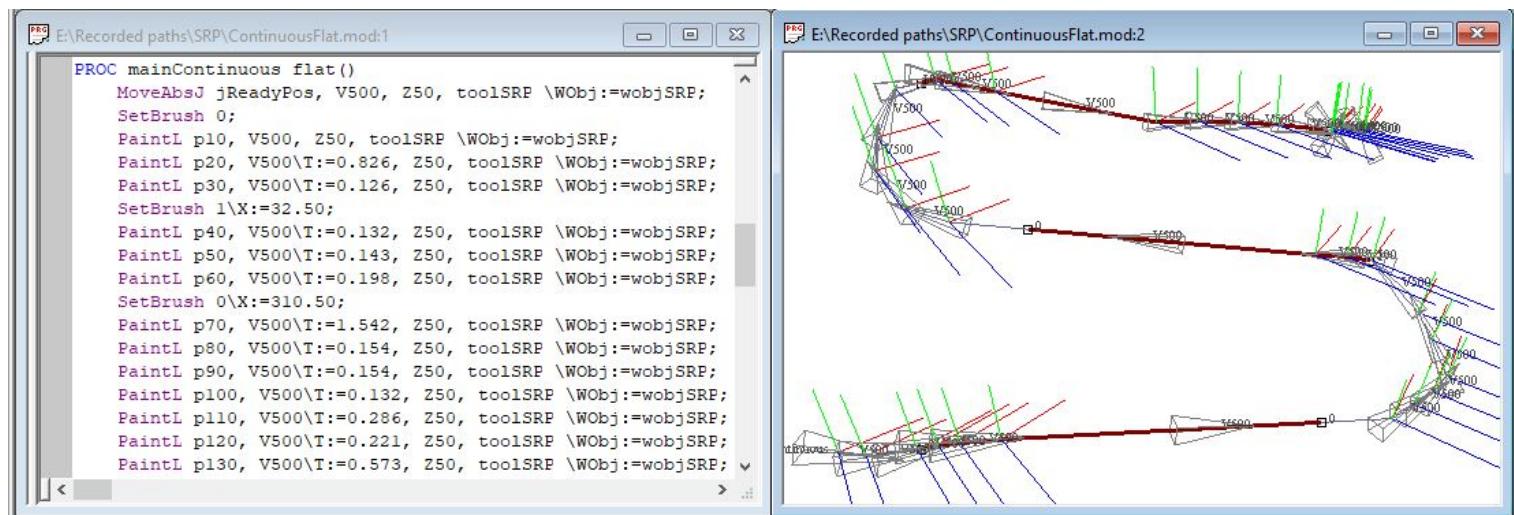


Figure 25: Continuous recording of a flat surface.

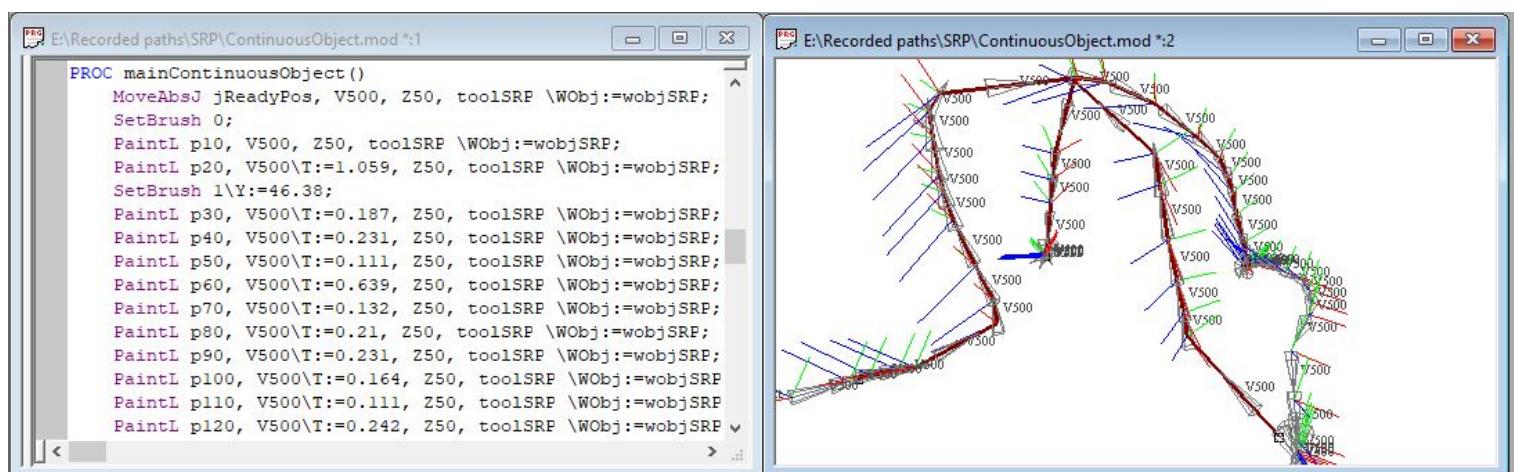


Figure 26: Continuous recording of a 3D object.

### 4.1.2 Point to point recording

For the first point to point path the same surface as for the continuous case is painted. The controller is moved along the path and the points are added by pressing the menu button. The trigger button is held when painting should be activated for that point.

The second path consists of painting a 3D box. Straight lines are recorded covering all visible sides of the box, first along one axis then along the axis perpendicular to the first. The two recorded paths can be seen in figures 27 and 28.

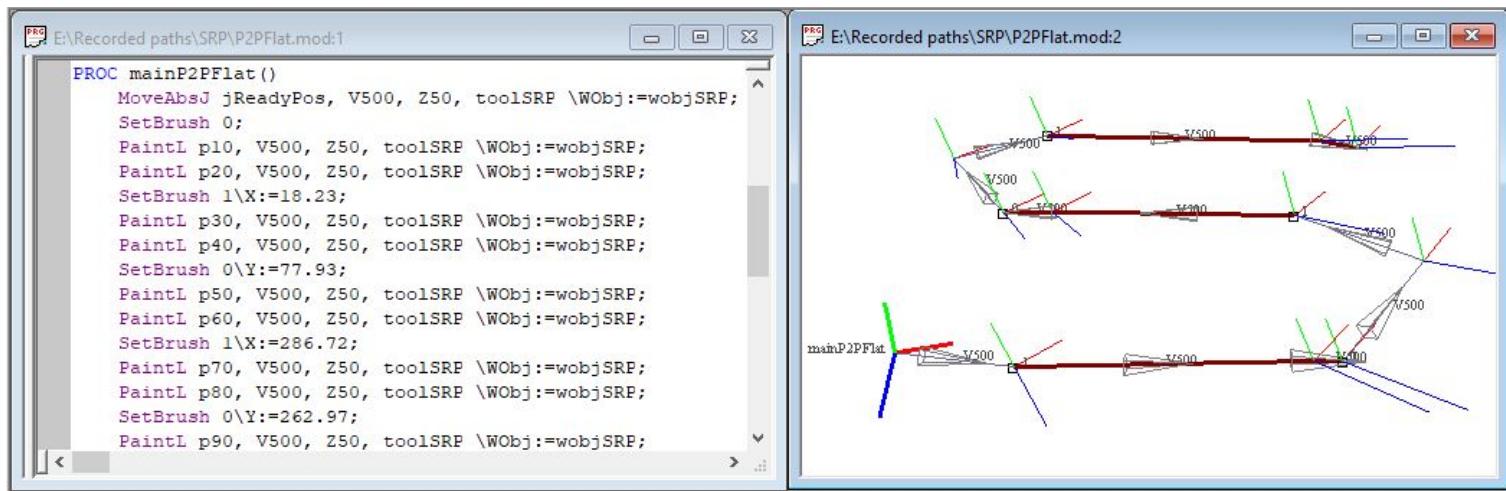


Figure 27: Point to point recording of a flat surface.

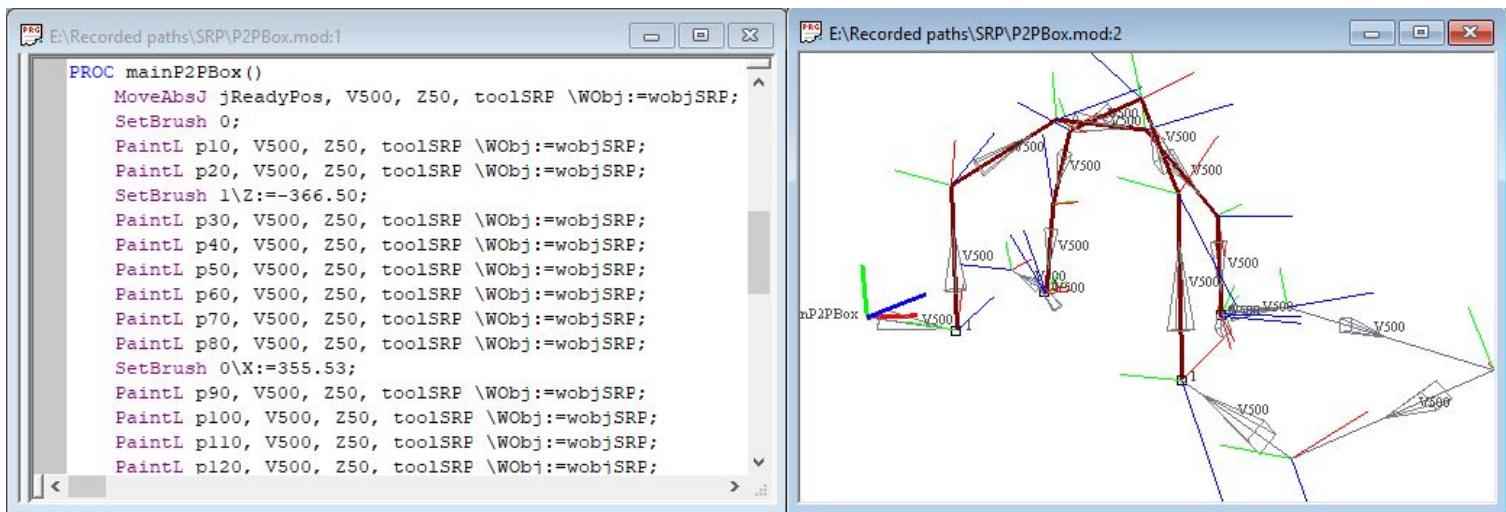


Figure 28: Point to point recording of a 3D object.

### 4.1.3 Reachability checker

To test the reachability checker, checkers with different parameters have been created. The hand controller is then moved in the positive and negative direction for all three axes to determine that the checker gives the correct output.

For the static box option a small cube is defined like in figure 29. The reachability checker gives green while the controller is within the given limits and red when it is outside. To check the work object offset it is tested with 200mm offset in each direction. The reachable area moves along the indicated direction as expected.

The static ellipsoid is tested in the same way. A few different checkers with varying parameters are tested. Moving along each axis the checker is red when the position is outside the outer ellipsoid, then turns green when entering the area between the two. It then turns red when inside the inner ellipsoid and back to green when it enters the area between the ellipsoids on the opposite side. This is true for each axis. Testing the offset of the work object is conducted like in the static box case with the same results.

Both of the implemented reachability checkers works as expected. It is also tested that recording is aborted if an unreachable point is added in the continuous case, and a point that is unreachable will not be added to the path in point to point mode.

## 4.2 Online Teaching

Online Teaching is tested by creating a new virtual IRB52 robot in RobotStudio, and the PC application is started in Visual Studio. A Vive controller and Lighthouse is connected through SteamVR and is tracking. The virtual controller is visible when the netscan window is opened and connection is successful. Through the described support functions the needed configurations and RAPID modules are automatically loaded onto the controller. After a controller restart the virtual robot is ready.

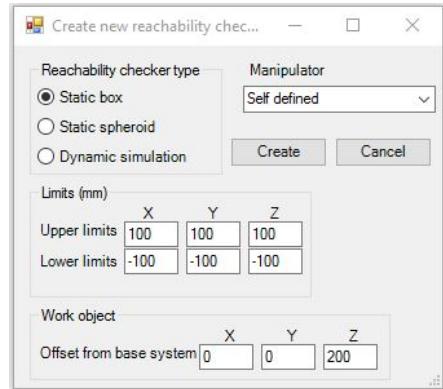


Figure 29: Creating different reachability checkers to test functionality.

#### 4.2.1 Continuous mode

To test continuous mode the corresponding GUI is opened. Upon opening rapid execution is automatically started and the robot moves to its designated initial position. The status of the sensor and robot are correctly displayed. To take control of the robot the grip button is pressed. The robot starts following the controller's movement, mirroring both position and orientation. Movement is smooth and no discernible delay between controller and robot can be seen. Pressing grip button again stops the robot from moving. To test correct calibration the controller is held with differing position and orientation while pressing the grip button to see that EGM movement can be started with an arbitrary pose on the controller.

Recording is tested by taking control of the robot and pressing menu button to start recording. The robot is moved to paint a flat surface similar to the first path recorded for SRP. After recording is done the recording is previewed on the robot and then saved as a text file. The text file is opened in SRP and the RAPID paint program in figure 30 is created.

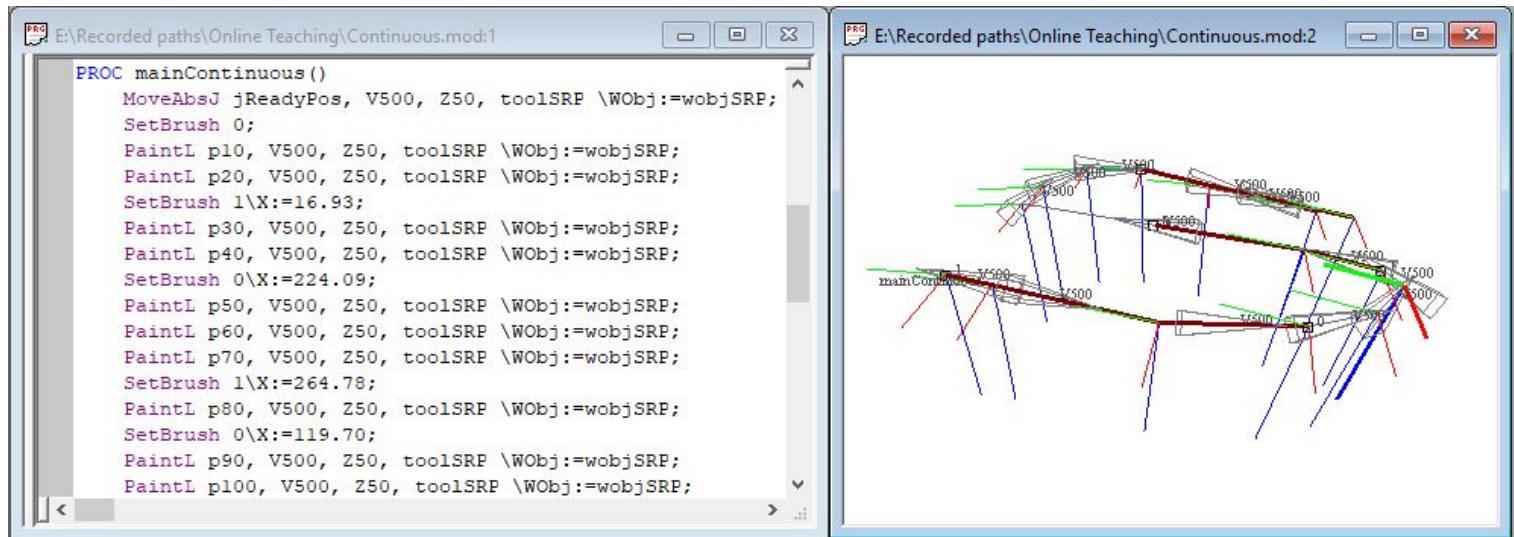


Figure 30: Continuous recording of a flat surface.

A test to record a 3D object is also tried. Recording a full path of an object similar to the second test for SRP is not successful, and the test fails. During recording the robot either runs into a constraint for one of the joints or encounters a singularity resulting in movement and program execution being stopped.

#### 4.2.2 Point to point mode

Similar tests are performed for the point to point mode. The preliminary tests for the GUI and taking control of the robot in different positions are carried out like in the continuous case with the same results.

A recording of the same flat surface as in the continuous case is made. Points are added by moving the robot to the wanted position and pressing up on the trackpad. The trigger button is held when paint should be turned on and released when it should be turned off. When all the points are recorded it is previewed on the robot and saved like in the continuous case, see figure 31. Using the right and left trackpad buttons the robot is cycled through all the recorded points. Deleting points is also tested by pressing down on the trackpad.

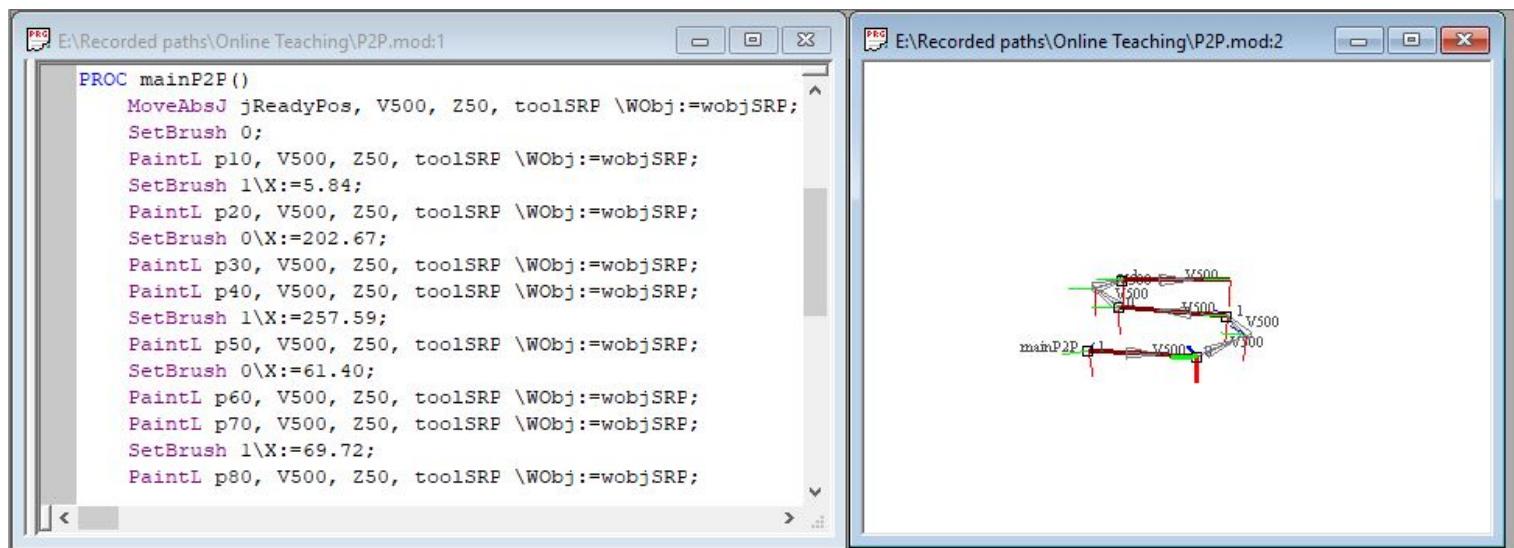


Figure 31: Point to point recording of a flat surface.

Similarly to the continuous case a test to record a 3D box is performed. Again the test fails due the robot encountering singularities or joint constraints.

# 5 Discussion

This section will discuss the implemented solutions and results of performed tests. It will highlight important weaknesses and propose possible improvements that could be made to the created systems.

## 5.1 Hardware

### 5.1.1 Sensor system

The HTC Vive as a motion tracking system has as expected performed well in this project. It was chosen largely because of good experiences with using the system in the past. Some considerations about the system are presented in the following section.

The precision of the system has proven adequate for use with paint robots. The speed of the robot while painting generally has the largest impact on the quality of the paint job. Errors on the magnitude of a few mm in the recorded path are insignificant. The largest discrepancy between the recorded path and the theoretical optimal path will in most cases be due to the human aspect.

One of the greatest strengths of the HTC Vive is that it uses infrared light for tracking. This makes it less prone to disturbances in the tracking area. The main issue with this approach is if line of sight between controller and Lighthouse is broken. This could occur when tracing large 3D objects while recording. When using only one Lighthouse this did happen on occasion, making the recording corrupt. However, this was never an issue when using two Lighthouses, which is the recommended setup. A new and better base station is also being released this year [15]. This will include the possibility of having more than two base stations in a room setup, further increasing the trackable area and reducing problems with occlusion.

HTC Vive also has a comfortable price tag compared to other solutions specialized for the corporate market. For small or medium sized businesses, which is the main target group for the software, the cheaper hardware could be a deciding factor in acquiring the system.

### 5.1.2 Controller

The HTC Vive controller was chosen as the HMI for this project. This was largely because of the familiarity to using the controller, and to free up time to spend on software development. However, from the viewpoint of the operator

using the system this is not necessarily the best choice. Some pros and cons of the three stipulated choices are discussed in the following.

### **HTC Vive controller**

One advantage of using the HTC Vive controller is that it is mass produced and readily available. It can be bought either directly from the HTC store or from most electronics retailers. As the popularity of Virtual Reality is rising and HTC Vive already has many interesting uses, many of the target group for this software may already own an HTC Vive set and no extra hardware is necessary. The Vive controller has a good amount of buttons that are versatile and well placed on the controller for ease of use. It also has a useful vibrating function that enables the program to give feedback to the operator without needing to see the computer screen. Using the Vive controller in addition makes the software more streamlined, as both the position tracking and buttons are handled by the same ViveTracker interface.

One weakness of the controller is that it does not resemble a paint gun. For an experienced painter the HMI looking and handling like a standard paint gun could make the transition to recording paint programs easier. Adding custom add-ons is also complicated when using the controller and could require modifications that would void the warranty.

### **SRP Teach Handle**

Using the existing teach handle would create little extra work. A mount for the smaller Vive Tracker would need to be made and, the position and orientation of the tracker relative to the tip of the teach handle must be found. This would make it possible for users of the existing system to upgrade to the new sensor by buying only the Lighthouses and Vive Tracker. The teach handle already has a laser light that is activated when the trigger is pressed. Making additional add-ons to the teach handle would be relatively simple.

The greatest weakness of the teach handle is that it has a limited amount of buttons, only the trigger and two function buttons. This limits extra support functionality that could be added. Having the ability to change brush number or paint color on the fly while recording is a nice possibility. The Vive Tracker being mounted on the teach handle would also make it bulkier and heavier. This could make it more cumbersome to use when recording paths.

### **Making a new HMI**

Making a new HMI would be a lot more work, but could be rewarding in the long term. An optimal place for the Vive Tracker mount could be made. The

amount of buttons and their placement could be optimized both in terms of needed functionality and ease of use. Having a mount for a phone or tablet on the hand held controller would create endless customization options. With a phone app connected to the software the user could customize what information and options to display on the phone.

The main disadvantage of this option is of course the increased development time. Making a proprietary tool that is produced in low volume might also increase the price of the hardware for the user.

## 5.2 ViveTracker interface

The implemented ViveTracker interface has worked well. It adds only one layer of code between the application using it and the OpenVR API. This makes making changes and adding new functionality simple. As it is being used by both SRP and Online Teaching the the complexity of the interface is kept as low as possible. Higher level functions like storing paths is not implemented. The segregation of the trackpad into four separate buttons gives more options to the developer.

Data is only updated when the `UpdateData()` function is called. This means that the system using the interface must utilize some form of timer or update loop to periodically update the state of the controller. If this update loop needs to run at a high rate it uses significant computer resources. To alleviate this issue an event base data update could be used. The OpenVR API includes a set of events that can be subscribed to. By using events for position and button state the ViveTracker interface could inform the higher level system as soon as a state change happens instead of waiting for the system to ask for new information. This could both increase responsiveness and reduce the computational resources used.

## 5.3 Simplified Robot Programming

As SRP is already an existing product it has a higher degree of finish than the other system. This makes implementing new functionality both simpler and harder at the same time. Although it already has frameworks for UI and error handling for example, a lot of time needs to be used to understand the existing code and how it interacts before changes can be made. All changes are made to be backwards compatible with the current sensor. This puts some limitations on what can be implemented, for example limiting the software to using three buttons.

Using the HTC Vive as the sensor has worked well. Its biggest improvement over the current sensor is that no precautions need to be made when painting metal objects. Adding support for the HTC Vive does not necessarily mean that the Polhemus Liberty system is obsolete. The choice of sensor system can be tailored to fit the needs of the user.

### 5.3.1 Recording mode

The performed tests show that creating paint programs is simple and efficient. Output from the paint program creation corresponds well to the path traced by the controller. The continuous flat surface recording shows the effect of the reducer algorithm. For the straight lines only a few points are needed to describe the path and redundant points are removed, while for the curved parts many points are needed. The RAPID code shows that the trigger points are detected correctly through the SetBrush commands.

The recording of a 3D object shows that more advanced paths can also be recorded. However, for parts of the recording the orientation seems to be slightly off. This could be due to the controller not being held perfectly perpendicular to the recorded path, but could also be due to the transformation of the Vive controller's reference frame. It is translated and rotated so that the origin is at the tip of the controller and the Z-axis points straight out of the tip. This transformation was found based on tests performed with the controller, as official information about the controller's coordinate axes could not be found. If this rotation is slightly off it could cause the orientation in the recording to be suboptimal.

The addition of the point to point recording mode makes the software more versatile. The biggest advantage is that the finished paint program has much fewer points. This makes editing the paint program after creation simpler. One drawback to the mode is that the speed of the robot can not be as easily controlled. Neither the time nor speed between recorded points is relevant in this mode, and the default speed in the program settings must be used when creating the paint program. This means that having different speeds for different segments is not possible. However, the speed can still be changed in the editing software afterwards, and with such few points in a recording this would be relatively simple.

### 5.3.2 Reachability checking

The reachability checking framework is working according to the specifications. It lets the operator create different reachability checkers and gives the correct

output and feedback to the user. The two implemented checkers has some merit, but also some glaring weaknesses that will be discussed here.

The static box checker's biggest advantage is its simplicity. It is very easy for the operator to create and understand what it signifies. The most relevant usage is for safety and space considerations. If the robot being used has a clearly defined area of legal operation, for example to avoid hitting walls or other objects in the working area, this area can be defined in the SRP plugin and it will give feedback if the path is within these safe limits. This can avoid dangerous situations if the recorded program is to be run directly on the robot after recording. The biggest weakness is that it does not take into account the orientation of the tool. Large portions of the manipulator arm can be outside the "safe" area even if the recorded TCP is inside. However, for simple paint operations like when flat objects are painted, the orientation of the tool stays mostly the same for the whole operation. These issues must be taken into consideration when discerning if the reachability checker can be trusted or not.

Static ellipsoid reachability checker is closely tied to the robot's working area. It is simple to define the inner and outer limits based on drawings of the robot's working area like in figure 13. Although it gives a good indication whether a point is reachable based on the geometry of the robot, this method also suffers from not taking orientation into account. Using the dexterous working area (the area the manipulator can reach with an arbitrary orientation of the tool[1]) instead would alleviate this issue somewhat, but would give a more restrictive reachability check.

To answer the main issue with both these methods a dynamic model reachability checker could be implemented. Developing such a dynamic model based on the kinematics of the robot is very time consuming, especially given the sheer amount of different robots ABB has in its lineup. However, ABB already have a framework for simulation of robot manipulators called RCS [26]. It is based on the Realistic Robot Simulation Interface Specification (RRS) [27]. After initializing a robot manipulator the user can send a position to the service and get feedback on whether it is reachable or not. It is not known how fast the response is and whether it would be possible to run in real-time. If running in real-time is not feasible, the whole path could be simulated using RCS after recording. This would still be simpler and save time compared to loading the program into RobotStudio to check reachability.

The suggested reachability checkers are not necessarily mutually exclusive in use. It would be possible to combine two or more checkers to run at the same time. For example a static box checker could be used to account for safety or space restrictions while an ellipsoid checker determines if the recorded path is

actually inside the robot's working area.

### 5.3.3 Improvements and future work

If an object consists of both straight and curved surfaces, being able to change recording mode on the fly would be intriguing. The straight surfaces could be recorded with straight lines using point to point, while the curved surfaces would be recorded using continuous motion. This would get the best of both worlds from the two recording modes. For straight lines it gets the added precision and simplicity of point to point while still being able to record curved paths. This would require a button on the controller to switch mode and would be harder to implement with the SRP Teach Handle. An indicator on the controller showing the current recording mode would also be helpful to the operator.

The creation of the paint program based on the recorded path could also be optimized. For example the speed does not need to be the same throughout the whole recording. While painting, the speed of the robot is important to the thickness of the applied paint, but not when paint is triggered off. A higher speed could be used when the robot is moving without painting. This would reduce cycle time and could increase productivity.

Having the ability to change program settings directly from the controller would also be a useful addition. Changing parameters such as paint color or brush number without needing to access the PC application could speed up programming even further. The controller could be customizable by mapping different functionality to the trackpad buttons depending on the user's needs.

## 5.4 Online Teaching

The tests performed show that the Online Teaching system is working but has some significant weaknesses. A lot of work is needed before it can be considered ready. The main selling point of the system is that it allows the operator to paint the object in real-time while the path is being recorded. This gives immediate feedback about the quality of the paint job and that the recorded path is reachable by the robot. This could save a significant amount of time compared to other methods of robot teaching if the first recorded paths does not give a satisfactory result. It works both with virtual and real robots, but controlling the robot by looking at the screen is awkward. The system is mainly designed to be used with a real robot. If programming is to be done offline (without the robot) SRP is a more natural choice.

Controlling the robot with the controller feels simple and intuitive. Little practice is needed before it feels like second nature, although finely maneuvering

the robot to a given position and orientation can be challenging. Controlling the robot translates well to paint operations where the speed and fluidness of the motion is more important to get a good result than millimeter precision. Although there is a slight delay between the controller and robot's movement, it is small enough that to the operator it feels like the robot follows the controller in real-time.

The system is targeted towards creating simple paint programs for prototyping and low volume production. Therefore the ease of installation and use of the system is important. A typical user might not have extensive experience with RobotStudio or ABB robots. All the necessary configurations are done automatically. As of now the only thing that must be done manually when setting the system up with a new controller is to define the work object and start position in the RAPID code. It should be possible to also automate this by telling the user to move the robot to the preferred initial position before starting the configuration procedure. The program could then get the current position from the robot directly and write it into the RAPID code. Then the user only needs to install the Lighthouses to give good coverage of the teaching area and set up the work object before the system is ready to start recording.

A lot of effort have not been put into UI design, error handling and adding setting options. The focus have been to make the system work and show the possibilities with real-time teaching. It is also cumbersome to first save recordings as a text file which needs to be opened in SRP to create the RAPID paint program. If the Online Teaching system is to be developed further to make it into a finished product however, it would most likely be integrated as a plugin to existing ABB software like RobView or RobotStudio. Then it would have access to a plethora of settings, options and functionality directly.

#### 5.4.1 Calibration and recording

##### Calibration

Calibration of the position part of the pose works well. When EGM movement is initiated a reference point is set and new positions are calculated relative to this point. Since both the controller and robot position is used when calibrating, both the controller and robot can be in arbitrary positions when EGM movement is activated. This is especially important for the point to point recording mode. If the operator uses the buttons to check previous recorded points, the robot will move to these points. Then when the operator wants to take control of the robot again it could be in a different position from when the previous EGM movement ended.

Orientation calibration have proven to be more challenging. The chosen

implemented solution works well when a tool with no rotation relative to the wrist center point is used. When the robot is moving it rotates around the same axes as the controller. This is not the case when a tool with a rotational element is introduced. The calibration is still correct when the button to take control is pressed, it does not start changing orientation immediately. However the axes of rotation does not correspond correctly between the controller and the robot. This is very confusing for the operator and makes it nearly impossible to record paths with changing orientations. Adding the rotation quaternion of the tool definition into the calibration equation has been tried, but this only make matters worse. Then the robot moves to a new orientation immediately when the button to take control is pressed even though the controller is held still. This problem is a serious issue that needs to be worked out, as for most paint gun tools the TCP is rotated relative to the wrist. With the current system it works best when flat surfaces are painted and the orientation does not need to change.

## Recording

The performed tests show that recording works well when painting a flat surface. Both the continuous and point to point mode show promising paint programs. The position and orientation is easily seen from the graphical view. Both show three straight lines perpendicular to the painted surface and the orientation of the tool is kept the same for the entire recording. Correct triggering of paint application can be seen in the RAPID code with the SetBrush instructions turning paint on and off. Again, the main difference between the two modes is the amount of points. If the program needs to be edited after creation, fewer points is preferred.

When trying to create a more advanced paint program of a 3D object, the system failed. The test was performed many times but was never able to complete a full recording. Almost every test failed due to encountering a singularity. That the solution would be prone to the robot encountering singularities was an expected issue. Since EGM bypasses path planning it has no way of choosing a configuration when a singular point is reached. If the robot encounters a singularity during an EGM movement an error occurs and rapid execution is halted. The cause of a singularity is often because of the wrist configuration. Only being able to create paint programs for flat surfaces is a limitation that severely reduces the usefulness of the system. A few possible solutions are discussed in the next section.

### 5.4.2 Possible solutions to singularity problems

#### Using a more advanced robot

Many ABB paint robots use a more advanced wrist configuration called a hollow wrist. This is a wrist made specifically for paint robots. It is hollow in the middle so that paint hoses can be brought to the paint gun inside the wrist instead of being fastened to the outside of the robot. This wrist configuration is also more robust and is less prone to experiencing singularities. However, from testing performed it seems that robots with hollow wrists are not compatible with EGM. When an EGM movement was started on these robots, the robot would start drifting off uncontrollably even though the position reference sent to the EGM module was constant. This was the same for several types of robots like IRB5500 and IRB5400, both live and virtual. This problem was never encountered on a robot without the hollow wrist configuration and it is assumed that the hollow wrist is not compatible with EGM in its current state. Until this is fixed, using robots with hollow wrists with the Online Teaching software is not feasible. This severely limits the usefulness of the system as a large amount of robots used in industrial painting processes uses this wrist.

#### Monitoring kinematics

Another approach to avoid singularities would be to check the position reference for singularities before it is sent to the robot. This could be implemented in the software itself by using the kinematics of the connected manipulator, or by using a framework like the RCS described earlier. As EGM is running at a rate of up to 250Hz this requires the position check to be very fast. If a too large delay is introduced between the controller's and the robot's movement it would be harder for the operator to control the robot. One way to alleviate the strict time requirement would be to predict the controller's future position. The position can then be checked for singularity before it is time to send it to the robot.

#### Hybrid approach

If the problems with singularities are not possible to solve in real-time at the moment, an approach similar to SRP could be used. The software is still connected to the robot, but the robot is not controlled directly with the controller. The path is recorded similar as for SRP, but can be instantly run on the robot through the preview procedure in Online Teaching. It would even be possible to record the path in segments, checking the quality of each segment before moving on. This would not need the use of EGM, and will not suffer the limitations this introduces. For the point to point mode the robot could move automatically to

a new point when it was added. This gives a semi-real-time approach to robot teaching.

### 5.4.3 Improvements and future work

An option to "lock" the robot on a given axis or in a given orientation could be added. If painting a flat surface the distance from the tool to the object and the orientation of the paint gun does not need to change through the entire recording. The robot is first brought to the starting location. By pressing a button the orientation and one axis is locked and the robot can only be moved in the plane parallel to the object to be painted. This could possibly also be stored as a setting if flat surfaces of different sizes need to be painted. Keeping the orientation locked should also reduce the probability of encountering singularities.

It would also be possible to add support for other sensors. If a Virtual Reality system compatible with the OpenVR API was to be added, minimal changes to the ViveTracker interface would be needed. The rest of the software is invariant to what sensor is being used. Adding support for the Polhemus Liberty would let users currently using SRP adopt the new Online Teaching without needing any new hardware.

Integrating the Online Teaching system into existing ABB software would also be a reasonable future development goal. A natural choice would be RobView, either as a part of SRP or as a separate plugin.

## 6 Conclusion

The purpose of this project was to develop an intuitive, efficient and simple way of teaching paint robots. This was done by using motion tracking to capture the movements of the human hand while simulating painting an object. The virtual reality system HTC Vive has been used for the motion tracking, and a software interface to interact with the hardware has been implemented. The HTC Vive system has performed as desired and is a viable solution for the intended use.

The project has been comprised of two tasks with a roughly equal amount of focus. Both tasks seek to implement a system for teaching paint robots using motion tracking, but with slightly different approaches.

The first task consisted of expanding the existing ABB software Simplified Robot Programming. This system lets the user create paint programs by tracing a path with a hand held device. It has been adapted to use the HTC Vive as its sensor, and new functionality in terms of a new recording mode and reachability checking has been added. Tests of the system show promising results and the system shows a high level of completeness. Little work would be necessary to make it ready for release as a finished product and the task is deemed completed satisfactory.

The system Online Teaching has been created to answer the second task. It lets the user control an ABB paint robot with the hand held Vive controller to paint an object in real-time. The path of the robot while painting is recorded and used to create the paint program. Tests show that simple paint programs for flat surfaces can be created, but the solution has several serious issues. The main problems are with calibration of orientation between the Vive controller and robot, and the robot encountering singularities. This results in program error and halted movement while recording more advanced paths. Possible solutions to the given issues have been proposed. The system in its current state is seen as a prototype or proof of concept and the task is completed only in part. Much work would be necessary to make this usable for paint robot teaching.

## 7 References

- [1] Hutchinson Spong and Vidyasagar. *Robot Modeling and Control*. John Wiley & Sons, United States, 2006.
- [2] Robert A. Adams and Christopher Essex. *Calculus - A complete course*. Pearson Canada Inc, Toronto, Canada, 2010.
- [3] John Vince. *Quaternions for Computer Graphics*. Springer-Verlag, London, UK, 2011.
- [4] Gergely Vass. Avoiding gimbal lock. *Computer Graphics World*, 32, June 2009.
- [5] Kristian Sletten. Automated testing of industrial robots using htc vive for motion tracking. Master's thesis, University of Stavanger, 2017.
- [6] Eivind Haus and Kåre Reime. *3D Pantograf - Styring av ABB Robot med HTC Vive*, 2017.
- [7] Microsoft. *C# Guide*, 2017. <https://docs.microsoft.com/en-us/dotnet/csharp>.
- [8] Microsoft. *C# Reference*, 2017. <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/index>.
- [9] Ken Loveday. *Microsecond and Millisecond C# timer*, 2013. <https://www.codeproject.com/Articles/98346/Microsecond-and-Millisecond-NET-Timer>.
- [10] stackoverflow user: Mr.Wang from Next Door. Get local ip address, 2017. <https://stackoverflow.com/questions/6803073/get-local-ip-address>.
- [11] Quaternions, 2016. <https://quaternions.online/>.
- [12] Google. *Protocol Buffers*, 2017. <https://developers.google.com/protocol-buffers/docs/csharpTutorial>.
- [13] Google. *protobuf-csharp-port*, 2015. <https://github.com/jskeet/protobuf-csharp-port>.
- [14] Valve. *Open VR API Documentation*, 2015. <https://github.com/ValveSoftware/openvr/wiki/API-Documentation>.

- [15] Ben Lang. *SteamVR Tracking 2.0*, 2018. <https://www.roadtovr.com/steamvr-tracking-2-0-will-support-33x33-foot-playspaces-with-4-base-stations/>.
- [16] HTC Corporation. *Vive PRE User Guide*. United States, 2016. [https://www.htc.com/managed-assets/shared/desktop/vive/Vive\\_PRE\\_User\\_Guide.pdf](https://www.htc.com/managed-assets/shared/desktop/vive/Vive_PRE_User_Guide.pdf).
- [17] HTC Corporation. *HTC Vive Tracker - Developer Guidelines*. United States, 2017. Version 1,5.
- [18] Oliver Kreylos. *Lighthouse tracking examined*, 2016. <https://drive.google.com/drive/u/0/folders/0B4vrRe4YjiTRIViektrOVBRdWs>.
- [19] ABB AB Robotics. *RobotStudio SDK - API reference*, 2018. <http://developercenter.robotstudio.com/robotstudio>.
- [20] ABB AB Robotics. *PC SDK - API reference*, 2018. <http://developercenter.robotstudio.com/pcsdk>.
- [21] ABB AB Robotics. *PC SDK - Manuals*, 2018. <http://developercenter.robotstudio.com/pcsdk/manuals?Url=html>
- [22] ABB AB Robotics. *Application manual - Controller software IRC5*. Västerås, Sweden, 2018. Document ID: 3HAC050798-001, revision: G.
- [23] ABB AB Robotics. *Technical reference manual - RAPID Instructions, Functions and Data types*. Västerås, Sweden, 2018. Document ID: 3HAC050917-001, revision: G.
- [24] ABB AB Robotics. *Technical reference manual - RAPID overview*. Västerås, Sweden, 2018. Document ID: 3HAC050947-001, revision: G.
- [25] ABB AB Robotics. *Technical reference manual - RCS*. Västerås, Sweden, 2016. Document ID: 3HAC048817-001, revision: C.
- [26] ABB AB Robotics. *User's guide - RCS*. Västerås, Sweden, 2017. Document ID: 3HAC048818-001, revision: C.
- [27] G.Schreck R. Bernhardt and C. Willnow. Realistic robot simulation. *Computing and Control Engineering Journal*, 6, August 1995.

- [28] ABB AB Robotics. *Product manual - IRB 52*. Bryne, Norway, 2017. Document ID: 3HNA011253-001, revision: 19.
- [29] ABB AB Robotics. *RobTracker Documentation*. Bryne, Norway, 2017.
- [30] ABB AB Robotics. *Trouble Shooting Manual*. Bryne, Norway, 2017.
- [31] ABB AB Robotics. *Product manual - Simplified Robot Programming*. Bryne, Norway, 2016. Document ID: 3HNA023965-001, revision: 3.
- [32] ABB AB Robotics. *Simplified Robot Programming - Data sheet*, 2015.

# Appendices

## A Online Teaching source code

