Robot Control and Computer Vision for Automated Test System on Touch Display Products

Ragnar Wernersson

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
ISRN LUTFD2/TFRT--5967--SE
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Media-Tryck
Lund 2015

Abstract

The goal of this master thesis is to set up a low cost automated robotic test system which can later be reproduced to greatly increase test coverage. Mostly through experimental research this thesis will find good components to use and evaluate these from a performance and cost perspective. It will also develop computer vision algorithms needed and automation software for the final set up. The performance of the robotics will be investigated by comparing with an industrial robot.

A modified 6000 SEK 3D-printer was selected and proved to work. The computer vision was developed using OpenCV and a fully automated system with appropriate resulting plots was created. Comparing with the industrial robot the setup using a 3D-printer proved to work better because it allowed for better positioning of the camera. It was also concluded that the selected robot system based on a 3D-printer was capable enough and would drastically lower the space and cost from a system using an industrial robot.

Acknowledgments

These people helped me a lot with my work.

Magnus Midholt, Katja Sjögren, Magnus Franzon, Marcus Numminen, Johan Erlandsson, Björn Ståhl, Emma Andersén, Eva Ståhl Wernersson, Håkan Håkansson and Anders Robertsson.

Contents

1

Problem Formulation

This thesis will investigate the possibility to set up low-cost fully automated test system for testing performance on touch display products. The goal is to increase the test coverage but also to detect issues in new software that can cause new bugs.

The task will be to find a low cost robot which will be connected with a high frame rate camera and a computer. The robot will perform swipes and computer vision will be used to track and visualize data.

After a working setup has been created the tests should be compared to tests on a more expensive professional robot.

2

Introduction

This master thesis was announced by the Device Driver team at Sony mobile. Sony Mobile operates on a highly competitive market with a large product line of mobile devices. All of these devices often consist of slightly different hardware builds within the models them self and have various software branches. These branches can vary in between both countries and operators and updates are pushed daily. In such an environment it is very important with good testing performance and quick feedback. To be able to verify and test all of this, good automation is needed for testing and presentation of measurements.

An automated test system will be set up which will measure time-delays in swipe moves using a robot for swiping and a camera for collecting input. Computer vision will then analyze the input and will be used to visualize the data.

By using a robot it has the benefit of actually testing on the final product similarly to the user experience. Many automated tests are done at a software basis but that does not, in any way, guarantee that the outcome will be the same on the final hardware. The camera provides an external measurement source so that the collection of data does not interfere with the performance of the tested product itself.

A few requirements has been set out by Sony; the first being that the system should be able to handle swipes from 40-400 mm/s on a variety of display sizes. The second requirement is to keep as low cost as possible. This basically means that professional robots are out of the question and that the software should stick to open source as much as possible. Test systems using professional robots are often very expensive and can still usually only perform tests on one product at the time. Using low cost, smaller robots, will allow for many systems working in parallel instead.

A final goal is to have all of these system connected to Sony's server-based testing frame-work. This way, the test jobs can be triggered automatically on new software builds, which will give immediate feedback to the developer.

3

A brief history of 3D-printers

3D-printing was first discovered in 1984. It was discovered by Charles Hull and used UV light the cure photo polymers layer by layer by layer to create a 3D object. This method is called stereolithography. Two years after this discovery Hull started the company 3D systems which still operates. 3D systems released the first commercial printer in 1992. In these earlier stages of 3D printing the results were not sturdy enough to be used as final products but more often used as prototypes [*National Inventors Hall of Fame* Accessed: 2015-03-25].

As improvements where made during the 90s to improve the output quality, the application also expanded into other areas and in 1999 the first organ was created with the help of 3D printing. In 2002 a functional kidney was printed and 2010 scientists were able to print functioning blood vessels [Harris, Accessed: 2015-03-25].

Companies have of late also started to adapt 3D printing for creating advanced metal components. GE has recently started to develop fuel injectors for their jet engines using 3D-printing. To do this a cobalt-chrome powder is applied layer by layer [Freedman, 2011]. In Lund, Sweden, Arcam is a company that applies 3D-printing with metal components for both orthopedic and aerospace technology [*Arcam AB* Accessed: 2015-03-25].

As of late years 3D-printing has expanded massively for commercial use because the prices have dropped drastically these last years. 2007 was seen as a breaking point for 3D-printers when the first printer for a price below $10,000 was introduced to the market by 3D Systems. Today 3D-printers can be found for as low as $100 [*Peachy Printer* Accessed: 2015-03-25]. The low-cost revolution was said to have been started by Reprap which is a community project for free software using 3D-printing [*reprap.org* Accessed: 2015-03-25]. This has also created a entirely new hobby which has spread very fast. At the time of this report (January, 2015) there are 192 live projects involving 3D-printing at the crowd-funding community Kickstarter [*Kickstarter* Accessed: 2015-03-25]. This is the reason for why a 3D-

printer is the central tool for this master thesis it is because it provides an affordable open-source tool.

4

Method

4.1 Selection and modification

The cost and limitations need to be carefully evaluated for each component in the lab setup. The time needed for each part of the project also needs to be considered as there is a time limit for the thesis and many components need to be put together. The amount of modifications needed can for example effect both time and cost. For a robot and a camera these parameters are quite quantified but it is also important to consider when selecting software to use. As an example; open-source is free but might be less optimized or suffer from more bugs. When selecting software, the effect on dependencies is also good to consider. It is often possible to find downloadable solutions but when ever a new dependency id added, more work is required when setting up new systems and more issues can occur when these conflict or get outdated.

4.2 Development

When the equipment and software to use have been selected, a development strategy has to be set up. First, what is required from each component to achieve the final result and if they live up to these requirements need to be considered. In the hardware case, if the product specifications does not meet the requirements, then some modifications and validation should be done at an early stage to investigate if it is possible to improve the performance. Sony want swipes up to 400 mm/s. However, it is most likely not the velocity but rather the acceleration that can prove to be an issue. If the display is small a high acceleration will be needed to be able to reach the desired velocity. A larger display may not need as much acceleration but will probably be heavier and the added weight will put more pressure on the robots motors to accelerate.

The second step will be to consider adding limitations for the user to prevent damage or other issues when other users start to use the system. For the robot this can be limitations for the user to avoid performing dangerous moves or push the

robot too much. For the computer vision software some sort of requirement on the set up and how much noise which will be possible might be of interest.

During development, some effort should be made to allow for adding of new tests. This means creating a class structure which allow for new tests to be added without having to change the source code in any significant way.

Testing during development

During development a lot of testing is done continuously to make sure the system remains robust and that nothing has been broken. The robot will also need some sort of identification testing to investigate the limitations of the robot.

4.3 Automation and verification

When each part has been developed and tested they all need to be integrated to an automated system. What this means is deciding how proper handshaking between the different components should work. Once proper automation is established, extensive test runs need to be done to ensure robustness.

There is a professional Epson robot at Sony Mobile, seen in Fig. 5.2. This robot cost about 150,000 SEK and will be used as comparison with the less expensive system. The use of this robot implements the requirement of a more versatile computer vision software since it should handle recordings from different robotic systems. The results of the two robots should be compared and the advantages and disadvantages should be summarized.

The final verification will investigate the measurement differences inside the same system by running multiple swipes on the same product and robot. And it will compare this with how much the measurements differ between the two robots.

5

Selection and Modification

5.1 Robot

Introduction

For robot selection the focus will be on low cost 3D-printers by searching web-stores, kick-starters and forums as well as communicating with the department at LTH. These printers can be found for very reasonable prices and should naturally, due to their constructed purpose, have high precision and be able to work for long periods of time. Many 3D models take hours to print and can have very small details.

Quantifying the requirements from Sony Mobile, the robot should fulfill the following criteria:

R 1: Be manufactured for a low cost.

R 2: Have good precision in XYZ-space.

R 3: Be able to run 40-400 mm/s velocities. This means, together with a decent acceleration at least in XY-plane.

R 4: Have long life-time.

R 5: Have working area large enough for 10" displays.

Final choice

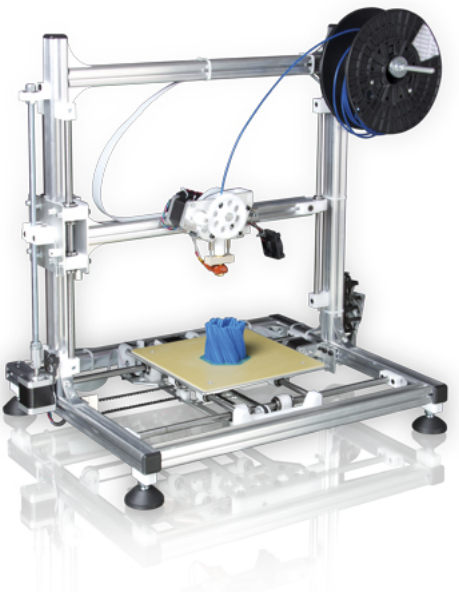


Figure 5.1 Vellman K8200 3D-printer [*Velleman* Accessed: 2015-03-25]

Name	K8200 3D-printer
Manufacturer	Vellman
Cost	5500 SEK (March 2015)
Stepper motors	4 Wantai 42BYGHW811 (NEMA 17)
Nominal mechanical resolution	X and Y: 0.015 mm Z: 0.781 μm
Printing speed	120 mm/s(typical) 300mm/s(maximum)
Max printing size	20cmx20cmx20cm

The choice of robot to use is a modified Velleman K8200 3D-printer seen in Fig 5.1. The printer has two stepper motors that moves a plate containing the sample, a third stepper motor moves the bar with the filament melter up and down and a fourth stepper motor to control the flow of the filament. The cost of a Vellleman 3D-printer is 5500 SEK (March 2015). The printer is delivered in parts, meaning there has been no pre-fitting of the parts what so ever. The cost per hour for building the printer argue against this specific model, but for this thesis it gives good knowledge of the product which can be an advantage when modifying it. The firmware for the controller card is open source and Arduino compatible [*Arduino* Accessed: 2015-03-25]. There is a support forum which has a lot of activity from both users and developers.

Comparing specifications to requirements

R 1: Fulfilled.

R 2: Fulfilled.

R 3: According to the specification 300 mm/s is maximum. This is below the requirements investigating through the community forum suggests that is should be possible to increase the voltage over the stepper motor controllers.

R 4: There is no information of estimated life time on these specific stepper motors. However, according to a different manufacturer a standard stepper motors lifetime is 10,000 hours [*Annheimautomation* Accessed: 2015-03-25]. A new stepper motor will then cost 249 SEK. Replacing of straps would cost 150:-. Replacing a stepper motor controllers would cost 150 SEK while a new controller board would cost 1300 SEK. It can be concluded that repairing the robot should be relatively cheap.

R 5: A 16:9 10" display would result in about a 22 cm long display. This is slightly too long for performing a full swipe. But it seem to be hard to find larger 3D-printer in the same price range. Most of the consumer 3D-printers found are smaller.

The fact that the K8200 uses an open source approach and that it seems to have a serious community is definitely also an advantage. Velleman markets the product as the "most hackable, tweakable and moddable 3D printer".

Comparing with the EPSON robot

Except for the price difference, there is a fundamental difference between the two robots in how movements are performed. As explained above, the 3D-printer moves the plate on which the sample is mounted while the arm is stationary. The Epson, see Fig 5.2, on the other hand, moves the arm itself and keeps the sample stationary. What this means is that the acceleration will, for the 3D-printer, be dependent on the weight of the sample while that is not the case for the Epson robot.

Figure 5.2 Epson robot

Other solutions

There are also 3D-printers based on paralleled kinematics, also known as delta-robot. These can be bought for about the same price using similar stepper motors and controller board. It will operate similar to the Epson where the tool is moved and the product is stationary which would allow faster movement because less weight is put on the motors.

Constructing

The printer was constructed according to the manual at first, with no modifications [*Velleman* Accessed: 2015-03-25], even though the actual 3D-printing properties were not going to be needed. The reason for this was to test the printer, knowing that it was working properly before starting to modify it. A print was made of "R2-D2s right leg" to establish that it worked properly, see Fig 5.3. When this had been confirmed the printer could be modified to fit the needs of this project.

It took about a week to build the printer. However, the next printer could probably be built modified straight away which would leave out some of the more advanced features.

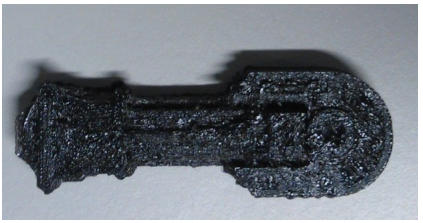


Figure 5.3 First and only print. R2-D2s leg.

Parts are that are not needed include the extruder with stepper motor, heating plate and fan. These are some of the more advanced parts and building time would probably be less the second time since these could be excluded straight away.

Mechanical modifications

The whole extruder part is removed. This includes the filament melter, fan and stepper motor. It leaves the top bar without any device. Instead a Z formed custom made aluminum part is mounted, see Fig 5.4. At the end there is a hole where a metallic cylinder is places. The hole is slightly oval so that the cylinder will not spin. The cylinder works as swiping finger it can move freely up and down and will only have its own weight pressing towards the display.

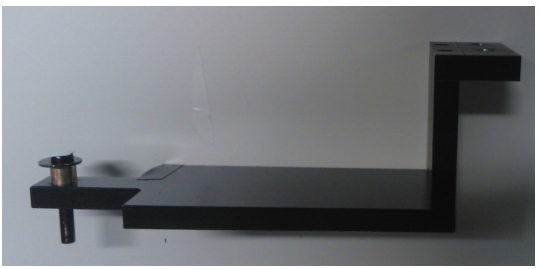


Figure 5.4 Arm for finger

The heater bed is removed and exchanged with a new fixture plate, Fig 5.5. The plate has incisions on the edges, parallel to each side, where three small blocks ($1cm^3$) are fixed. Two are put on the long side and one on the short side of the phone. It also has a long block compensated with two smaller blocks to tighten the phone from the other side. This design allows a flexible mounting of the product and provides easy connectivity of USB and other input cables. Since the location of buttons and USB connector varies in between models it is important to have this flexible mounting.

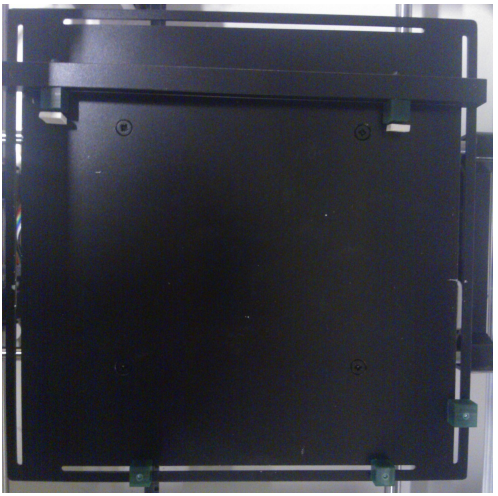


Figure 5.5 Fixture to fasten product

Software modification

Information of how to update the firmware as well as a download for the firmware can be found in the manual [*Velleman* Accessed: 2015-03-25]. There is also a git repository for the firmware [*Marlin* Accessed: 2015-03-25], but it should be kept in mind that changes may exist between the official Git and the download from Velleman, since it is under a GPL license. The printer uses an Arduino board called Marlin and requires the arduino software from [*Arduino* Accessed: 2015-03-25] to upload new firmware to the printer.

Most of the basic modifications can be found in the Configuration.h file. To be able to communicate at all with a Linux machine, the baudrate has to be changed from 250000 to 115200.

```
#define BAUDRATE 115200
```

When the filament melter is removed the file Temperature.cpp has to be modified. Since the printer is programed to do temperature tests these have to be disabled. The following code is commented.

```
    for(unsigned char e = 0; e < EXTRUDERS; e++) {
        if(current_raw[e] >= maxttemp[e]) {
            target_raw[e] = 0;
            max_temp_error(e);
            #ifndef BOGUS_TEMPERATURE_FAILSAFE_OVERRIDE
            {
              Stop();
            }
            #endif
        }
```

```
        if(current_raw[e] <= minttemp[e]) {
            target_raw[e] = 0;
            min_temp_error(e);
            #ifndef BOGUS_TEMPERATURE_FAILSAFE_OVERRIDE
            {
              Stop();
            }
            #endif
        }
    }
#if defined(BED_MAXTEMP) && (HEATER_BED_PIN > -1)
    if(current_raw_bed >= bed_maxttemp) {
        target_raw_bed = 0;
        bed_max_temp_error();
        Stop();
    }
#endif
```

5.2 Camera

The camera selected by Sony is a Ximea MQ003CG-CM [*Ximea* Accessed: 2015-03-25]. It has VGA resolution, 500+ frames per second (fps), 1/3" CMOS, Global Shutter and color. The price is around 9,000 SEK.

A standard mobile display has 60 Hz update frequency which would result in about 8 points of measurement for each frame-update using 500 fps. This will give good time resolution and should leave room for the possibility of performing tests on displays with 120 Hz. A drawback for the camera could be its low image resolution. Another issue is that the camera does not come with any recording software.

5.3 Software

To keep with the goal of using low cost tools, OpenCV [*OpenCV* Accessed: 2015-03-25] is selected as the computer vision analyzing tool. OpenCV is free to use and have good online documentation. It has a big community which makes it easy to find guides, tips and ideas.

OpenCV has support for C, C++ and Python. This project will be written completely in C++ because of previous experiences and OpenCV support. To be able to print output and to visualize results in a good way, the program will store the resulting values in a .tex file and compile with pdflatex together with the pgfplots-package [*PGFPlots - A LaTeX package to create plots* Accessed: 2015-03-25].

6

Development

6.1 Final setup

The following images, Fig 6.1 and Fig. 6.2, will give a brief overview of how the final setup will look like. Since there are two different robots that will be used, it is important to understand the physical differences between these to better grasp some of the decisions made during the development phase.

Comparing the different setups

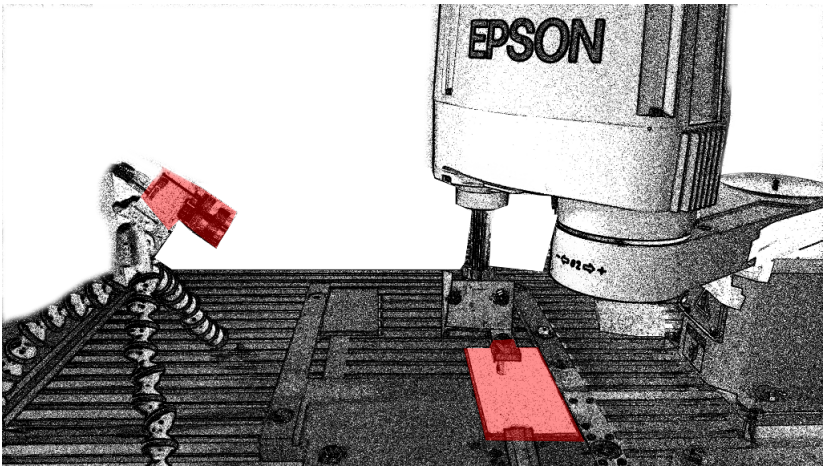


Figure 6.1 Epson robot set up. Touch finger, camera and phone have been marked red.

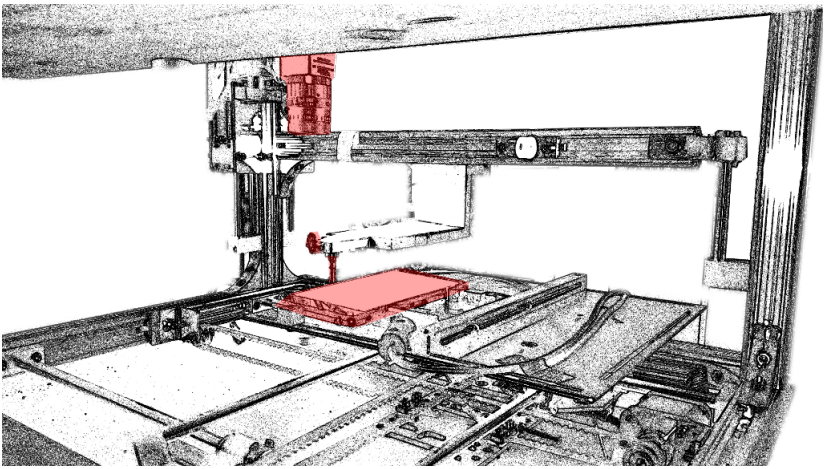


Figure 6.2 K8200 robot set up. Touch finger, camera and phone have been marked red.

The 3D printer moves the sample itself. Hence the amount of acceleration possible will be dependent on the weight of the product. This could prove to be an issue for heavier products such as tablets. This is not an issue on the Epson robot since it will instead move the swiping finger over a stationary product.

The benefits will be reversed when considering the camera. Unless the camera is mounted on the robot arm of the Epson, it will have to be positioned in a way so that the entire display is in view. This means that a larger display will require the camera to be positioned further away, in other words, worse resolution. The 3D-printer moves the product under a stationary finger and the area where the finger impacts is what is of interest for the camera. This means that the camera can be stationary and still get the entire swipe on the recording. The display will move from top to bottom through the view of the camera and so the entire swipe is covered with the same resolution independent on the size of the product.

6.2 Robot

Communicating with the printer

Stable communication is crucial for this project to work. The printer uses an open source Arduino-based controller-card called Marlin [*Marlin* Accessed: 2015-03-25] It uses a standard USB 2.0 cable for connection.

Behavior The printer sends and receives strings always ending with new-line `"\n"`. If the controller card doesn't recognize the command sent, it will reply with `"Unknown command"` and echo the string sent, after which it will continue reading.

For most commands, the printer will reply with an "ok\n" when the command has been added to the buffer. Commands like G28(go home) or M400(wait for moves to finish), also known as blocking commands, will send "ok\n" only when the command has finished executing. There are also special commands that will return unique results such as M114(get position). A list of commands can be found in the G-code wiki [*reprap.org* Accessed: 2015-03-25]

Communicating software The base communication class is simple. It provides the functions needed for communication but does not provide any constraints or other safety checks. The printer itself has mechanical stopping switches and a knowledge of its position so that it will not move outside its area. However, since it uses stepper motors it cannot actually really know its own position, it can for example be set astray by an external force.

The idea with the communication software is that another layer will be added on top of the base communication which will provide safety checks and constraints. The communication software is almost entirely C-code but for easier integration it is implemented as a C++ class. The communication software has been developed using "Serial Programming Guide for POSIX Operating Systems" [Sweet, 2010].

The termios class is used for serial communication. Properties and baud rate can be set for correct communication. In this case using line by line communication, also known as canonical mode, should be applied with echo disabled. Also baud rate should be set to 115200 on a Linux machine.

```
struct termios options;
...
//Get attributes
tcgetattr(fd,&options);
cfsetispeed(&options, B115200);
cfsetospeed(&options, B115200);
//Enable the receiver and set local mode...
options.c_cflag |= (CLOCAL | CREAD);
//Set the new options for the port
options.c_lflag |= (ICANON | ~ECHO | ECHOE);
options.c_cflag |= CS8;
//Set attributes
tcsetattr(fd, TCSANOW, &options);
...
```

The port should be opened as read and write. Reading is set to blocking mode using the following command:

```
fcntl(fd, F_SETFL, 0);
```

This will make sure that every read command will block until finished reading.

Sending strings to the printer is possible with write(PORT,CHAR*,CHARLENGTH).

```
int robot_communication::writeGCode(const char *gcode)
{
        int n = write(fd, gcode, strlen (gcode));
        if (n < 0)
```

```
        {
                fputs("write() failed!\n", stderr);
                return 0;
        }
        return 1;
}
```

To ensure proper communication a function will check every char sent from the printer. The function only returns when a "ok\n" has been read.

```
void robot_communication::waitForOk()
{
        char ch1[1];
        while(1)
        {
                read(fd,ch1,sizeof(ch1));
                    printf("%c",*ch1);
                if(*ch1 == 'o')
                {
                        read(fd,ch1,sizeof(ch1));
                        printf("%c",*ch1);
                        if(*ch1 == 'k')
                        {
                                read(fd,ch1,sizeof(ch1));
                                printf("%c",*ch1);
                                if(*ch1 == '\n')
                                {
                                        break;
                                }
                        }
                }
        }
}
```

While `writeGCode` and `waitForOk` are private functions, `command` is implemented as the main public communication function. The function writes the sent in string to ttyUSB, waits for OK and returns 1 if successful or 0 if not.

```
int robot_communication::command(const char *gcode)
{
        printf("%s", gcode );
        if(!writeGCode(gcode))
        {
                printf("Couldn't send %s to Marlin\n", gcode);
                return 0;
        }
        waitForOk();
        return 1;
}
```

A begin function opens up the port and sets initial properties such as units to millimeters, absolute positioning, feed-rate and acceleration.

Also different functions are created for easier direct control of the printer. These takes an input value of float or int format and transforms it to a cor-

rect G-code string. Similar functions implemented are `moveX`, `moveY`, `moveZ`, `moveXY`, `setFeedrate`, `setAccerleration`, `getPosition`. For example:

```
int robot_communication::moveX(float x)
{
        char *gcode, *temp_str;

        /*Get size of string*/
        int size = 5;
        int temp_i = x;
        size += getStrSizeOfInt(temp_i) + 4;
        /*Allocate memory*/
        gcode = char[size]

        /*Create G-code string*/
        strcpy (gcode,"G01 X");
        floatToString(x,&temp_str); //allocates memory and
            creates string from float with 2 decimals.
        strcat (gcode,temp_str);
        strcat (gcode,"\n");

        /*Free memory and send command to printer*/
        free(temp_str);
        int tmp = command(gcode);
        return tmp;
}
```

The destructor will close the port opened by the begin function.

Identification

There is no need for a full system identification of the robot since it comes with the product. It can, already from its original state, move accurately between coordinates in proper SI units and be given different velocities and accelerations. The original setup is however constructed for lower accelerations, it is limited to 3000 mm/s^2 in the firmware, so it is desired to investigate a heavier load. As high acceleration as possible is of interest in this project since swipes up to 400 mm/s^2 will be performed. Higher accelerations will put more load on the stepper-motors and since the K8200 moves the product the amount of acceleration possible will depend on the product's weight. A visualization of acceleration and deceleration time can be seen in Fig. 7.1. Also, since the X-motor carries the Y-motor, the maximum accelerations will differ depending on the direction of the movement. To evaluate the printer limits two test are performed:

- Acceleration test using different voltage and weights. See Fig. 7.1
- Acceleration test using final mechanical modifications with fixed voltage. See Fig. 7.2

These tests are explained in full in Chapter 7 (Experiments).

From the tabular data in these tests, a simple linear model is constructed to saturate the acceleration. Since the amount of acceleration possible differ in X- and Y-direction, trigonometry is used to calculate the maximum acceleration in each move.

Compensating for diagonal movements The acceleration tests performed lead to assume that some function might be needed to restrict the acceleration depending on the angle of the move. A swipe in straight X- or Y-direction would of course be capped by the maximum possible acceleration in that direction. However for moves that are performed diagonally the maximum acceleration should be decided by the angle to either axis.

Simple trigonometry tells us that the acceleration for vector $\ddot{v}$ in x- and y-direction is given by the projection from $\ddot{v}$ to the respective axis:

$$\begin{aligned} \ddot{x} &= cos(\theta) \cdot |\ddot{v}| \\ \ddot{y} &= sin(\theta) \cdot |\ddot{v}| \end{aligned} \tag{6.1}$$

By considering the maximum acceleration allowed on each axis, the maximum allowed total acceleration $\ddot{v}$ can be known. The following pseudo code describes the function.

```
dx = x_prev - x_curr
dy = y_prev - y_curr

get x_max_acc  //weight dependent
get y_max_acc  //weight dependent

sin_theta = |dx|/|dx+dy|
cos_theta = |dy|/|dx+dy|

acc_max_y_trig = y_max_acc/cos_theta
acc_max_x _trig= x_max_acc/sin_theta

if(acc_max_y_trig > acc_max_x_trig)
 return acc_max_x_trig
else
 return acc_max_y_trig
```

If the returned value is lower than the current acceleration used, the acceleration should be capped to the returned value.

A quick verification test is set up performing 10 backward and forward swipes over the entire working area. Each swipe with an increasing angle from being parallel to the x-axis to being parallel to the y-axis. The idea is to use the linear models created in the acceleration test together with trigonometry to, for every move, calculate the maximum allowed acceleration. That way printer would have a built in saturation function. This also means that the product weight would have to be included as an input. This test is not described in Chapter 7 (Experiments), as it is merely a simple verification of the above model.

Overlaying class

With the identifications made in the previous section an overlaying class called robot_free_move is created. This class extends the communication class and uses the constraints measured so that safe use can be guaranteed by sending in the product information during initiation.

Integrating the constraints The acceleration constraints are implemented in this class. By comparing previous with current coordinates the acceleration is capped depending on the angle as explained earlier in this chapter. Also there is a function to warn when going outside the area of the display. It also performs a few "safe Z-moves" to avoid the risk of ramming the finger into the product.

Free run The class has a function for direct control of the robot using keyboard input which can be used to hard code tests. It will respond to inputs for moving around alongside xyz-axes. The tests are created by storing points, feed-rate and acceleration changes and pauses. It returns a vector with the G-code corresponding to the test created. During the creation of these tests, it is possible to test run the stored coordinates and to remove or to regret added coordinates.

During the test creation mode the following operations are possible:

- Change step size. Sets the length of each move.
- Move in any direction the length set, XYZ.
- Move to specific coordinate.
- Store command, point/feed-rate/acceleration/pause.
- Remove stored command.
- Run stored commands.
- Quit and save created test.

Running tests All tests run through this class to guarantee that the limitations are followed and to minimize the risk of injuring the sample product.

6.3 Camera

Recording software

The recording software for the camera is written by Björn Ståhl at the CI team at Sony Mobile. It is a simple C-program which uses a library included with the camera. Inputs can be sent in to change resolution, input frame rate, output frame rate, length, gain and filename.

The program also provides a list of the time-stamps for when the shutter closes. This provides the exact time within millisecond precision of each frame.

Recording to RAM A lot of data needs to be stored very fast when recording in high frame rate. To avoid issues such as sudden jumps in the output file the video can be recorded straight to RAM. By using the Linux command mount, a disk is created on the RAM where the file can be stored. After recording to RAM-file, it can be copied to the hard drive. The specific command used to mount on RAM-disk on this project is:

```
sudo mount -t tmpfs -o size=2G tmpfs place/to/mount
```

A modern computer with a SSD disk is a good investment since it may relive the use of `mount` so that all files might be recorded to the hard-drive directly.

FIFO - pipelined communication The recording software used is standalone. Starting a new recording requires about a second for the camera to get ready. To get as optimal handshaking between the robot and the camera the code is modified by adding named pipes. This is implemented using methods described in "The Linux Programmer's Guide: 6.3 Named Pipes (FIFOs - First In First Out)" [Goldt et al., 1995]

After the robot is in position ready for its move the main program will open a FIFO file as `read` and wait for response before continuing. The recording software is then started which will open the FIFO file created as `write`. As soon as it is ready to start capturing images it will write to FIFO and the main program will continue with the robot moves.

The robot control and camera recording runs in different threads. After starting a swipe and a recording the threads are synchronized.

Calibration software

A program called `streamViewer` is included with the Ximea camera and can be used to get a live feed from the camera directly. This program is used for calibrating.

Support for Ximea cameras can be added when installing or upgrading OpenCV. This allows the user to open the camera as a live stream in OpenCV. This can be used to create more advanced calibration tools. There seem to be issues with sending instructions to the camera through OpenCV, so it is currently not an ideal alternative.

Since there is an already working live feed solution bundled with the product, adding OpenCV support would add even more dependencies to the project and have therefore been disregarded. If a more advanced calibration program is decided to be needed in the future, this could be an option.

Positioning and distortion

Positioning When using the 3D printer the camera is stationed above the finger as seen in Fig.6.2. The Epson setup has the camera positioned from the side with a slight downward tilt as seen in Fig.6.1

Distortions There are mainly two kind of distortions that effect the results. Perspective distortion and lens distortion. No algorithms are introduced to compen-

sate for distortions. Fig.6.3 shows that the effect on distances is noticeable but very small. The perspective distortion is mostly effected by the positioning of the camera, see Chapter 8 (Discussion) for thoughts regarding this.

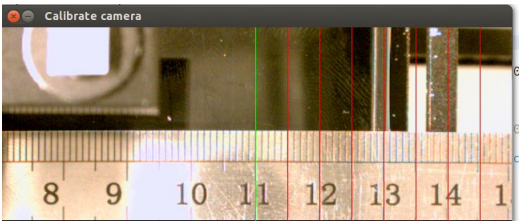


Figure 6.3 Lines are drawn with equal distance, green representing center of screen. The lines seem to follow the 5 mm pattern well, until the last centimetre.

Resolution

There are three aspects to look at. The camera resolution, the time-stepping resolution and the display resolution of the product. It is important to investigate which aspects will be the limiting factor depending on the setup. This is important because it is preferable if the camera is used in a way so that the quality of results are maximized. In Chapter 7 (Experiments) calculations are made on resolution and camera positioning. These results are verified by studying the quality of two different camera positions.

6.4 Computer Vision

Introduction

No real-time computer vision is required. The goal is to do as robust tracking as possible and the software should be able to handle the various physical differences between products (tablet versus smart watch). It should also work on both the Epson and the 3D-printer setup.

Brief history of the algorithm A first approach iterated through the video only once. It analyzed frame by frame, found the dots to track and also had an intelligence to sort the tracked dots. This ended up in a lot of constraints and if-statements to handle all the possible noise. Finally this approach was canceled.

A new approach is introduced where the measured input is evaluated a couple of times instead of per frame, which is unnecessary when a real-time approach is not needed.

The Android App

The purpose of this app is to provide an easy tracking of the display response and at the same time provide a solution of converting the distances from pixels to mm.

The app will print a rectangle on the display as soon as it senses a touch. The rectangle can be modified in size and offset from the touch point. It will also print reference dots on a distance of twice the offset from the touch point. The reference dots are constant in position and distance from each other and serves two purposes for the 3D-printer and one purpose for the Epson. The 3D-printer moves the product which means that the point of touch will always be centered. This means that only the relative distance for the object to track will be shown. By tracking the movements of reference dots the absolute distances can be calculated. The second purpose is to transform pixel lengths to SI units. The Epson uses a stationary product with a moving swiping finger and so only the second purpose is needed. An example of the app can be seen in Fig.6.6

In android it is possible to call Metrics [*Android, DisplayMetrics* Accessed: 2015-03-25] which provides information about the screen. In this app the dpi.x, dpi.y and screenHeight are of interest. By using the provided dpi everything that is drawn on screen can be converted to millimeters. This means that the distance between the reference dots will be the same on any product and by calculating these differences in the computer vision algorithm, a pixel to millimeter scaling transformation constant can be calculated.

During the rest of this thesis, the dot following the swiping finger will be referred to as the object dot.

Tracking

Since the 3D-printer has a stationary finger the camera doesn't need to track anything else than the dots on the display to work. Only tracking the movement of the reference dots and the object dot is needed. But tracking the robot finger is good to detect if the robot vibrates or moves during a swipe. This is solved by using a LED on the tracking finger. This also means that the robot can be put in an entirely dark environment and thereby reduce a lot of disturbances. The LED was added at a late phase and therefore a lot of example images in this report only have a rectangular white paper on the finger (very dark).

Inputs

The velocity of the swipe should be provided from the robot. This can be and is calculated in the computer vision algorithm but having the input provides good comparison and can prove as a good error checking. Since the program will always know the velocity sent to the robot it is logical to use this information.

Also the robot type needs to be sent in so that the algorithm knows if it is using absolute or relative movements, as explained in the app section above.

Another input is the video file and the corresponding time-steps. The time-stamp vector is used to convert frame counts to milliseconds.

From the app, the distance between the reference dots is needed as an input to get correct distance conversion.

The Program

Differences between Epson and K8200 The same app is used for both robots.

Code wise, there is very little difference in the tracking software and none what so ever in the post-processing software. Both robot needs the reference dots to calculate lengths, while the K8200 also uses them for absolute positioning. So, during the tracking, the K8200 version will not only store the mean distance value between reference dots but also the actual movement. Before sending the information to post-processing these movements are added to each point.

This is the reason to why the robot model is needed as an input to the computer vision program.

Image adjustments The image adjustments is based on the Canny function. A Gaussian filter is first added to smooth out small noise. Canny uses the Sobel operator and is applied to detect edges. By adding threshold levels it filters out unwanted edges outside these levels. Lastly OpenCV's `findContours` function is applied. It stores all points for each contour in a vector.

```
cv::Mat swipe_test::imageAdjustments(cv::Mat frame, cv::Rect
    ROI, vector<vector<cv::Point> > &contours, vector<cv::
    Vec4i> &hierarchy)
{
    contours.clear();
    hierarchy.clear();
    if(tech >1)
        tech = 0;
    if(tech == 0)
    {
        frame = turnGrayscale(frame);
        cv::GaussianBlur(frame,frame, cv::Size(3,3),5);
        cv::Canny(frame,frame,120,255,3);
        cv::imshow("Changed", frame);
        cv::findContours(frame, contours, hierarchy,
            CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, cv::Point(
            ROI.x, ROI.y) );
    }

    return frame;
}
```

Retrieve points The OpenCV function for bounding rectangles stores every vector of contours as a rectangle big enough to cover the respective contour. Rectangles that are either very small or have a width which is a lot larger or smaller than its height, are not stored (probably a line). This was developed using [*Createing Bounding boxes and circles for contours* Accessed: 2015-03-25]

```
/*
 * Detects bounding rectangles for each contour. Removes any
     contour that has a
 * bounding rectangle to small or to large compared to the
     tracked points.
```

```
 *       contours = stored contours
 *       boundRect = stored bounding rectangles
 *       constraints = if the function should remove to large
     or to small contours.
 */
void swipe_test::getRectangles(vector<vector<cv::Point> > &
    contours, vector<cv::Rect> &boundRect, bool constraints)
{
    if(contours.size() == 0)     //nothing is tracked
    {
        return;
    }
    for(size_t i = 0 ; i < contours.size(); ++i)
    {
        boundRect[i] = cv::boundingRect( cv::Mat(contours[i])
            );
    }
    if(constraints)
    {
        for(size_t i = 0; i < boundRect.size() ; ++i)    //
            remove to small or to large bounding rectangles.
        {
            float tmp = boundRect[i].height/(float)boundRect[i
                ].width;
            if(boundRect[i].height < 5  || boundRect[i].width
                < 5 || tmp < 0.25 || tmp > 4) //
            {
                contours.erase(contours.begin()+i);
                boundRect.erase(boundRect.begin()+i);
                --i;
            }
        }
    }
}
```

A rectangle in OpenCV is stored as a point representing the upper left corner coordinate and its width and height. The `getRectangle` function is performed in every frame. All the rectangles are stored in a vector with the same size as the amount of frames, each index contains the corresponding rectangles retrieved for that frame.

A global matrix and a global array is created. The matrix has the same dimensions as the width and height, in pixels, as the frame, each index representing a pixel coordinate. The array has the same dimension as a column of the matrix, where each index in this array represents the sum of each line in the matrix. Iterating through each frame, every rectangles found pixel position is stored in the matrix and in the line sum array by increasing that coordinate with 1. For example, a rectangle found in position (5,6) will result in matrix[5][6] += 1 and array[6] += 1.

After all the frames have been analyzed there will be a vector containing, for each frame, a vector with all rectangles found in that frame. There will also be a matrix containing the amount of points found in each respective pixel over the entire clip. And an array containing the sum of each point found in the respective

line over the entire clip.

Normalizing, removing noise and merging After the points have been retrieved, it has to be decided which points that are of interest. Knowing that all swipes are performed parallel to the robot's X-axis, areas with a high line-sum should be of interest. Having a high line-sum either means that it should be points to track or that there are stationary noise on those lines. Random noise should, on the other hand, be easy to avoid since they should not effect the line sum very much. The following steps are performed for selection and noise reduction.

The column vector containing the line-sums is normalized by dividing with the largest sum. All values less than 0.01, meaning a sum less than 1 percent of the largest, is set to zero and considered as noise.

Since the tracking dots covers more than one line, areas of interest are created by merging clusters of lines that have a high sum and lies next to each other.

Each area of interest is stored in a container with 5 values containing:

- Y-index of the highest line-sum in the respective area.
- The highest line-sum in the respective area.
- Start y-index of the area.
- End y-index of the area.
- Sum of all the line-sums in the area.

When there are lower line sums separating high line sums, they still might belong to the same dots. Selected areas that are very close to each other are compared pixel per pixel of their most dominating lines, similarly to XNOR operator. If they have a 99% similarity, they are merged.

The three areas with the highest total line-sum is selected. The following picture shows a swipe. Red crosses are tracked dots, green lines are selected areas, while purple lines are areas that are disregarded.

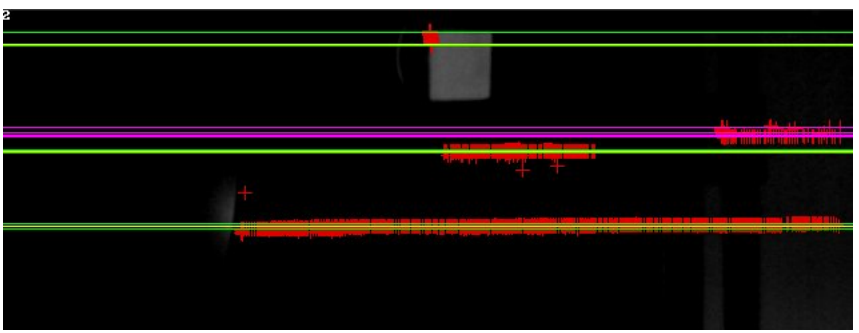


Figure 6.4 Before removing unwanted points. Green areas will be stored, purple will be discarded.

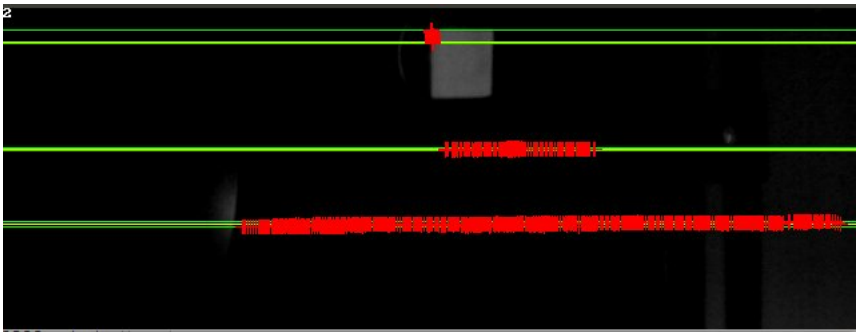


Figure 6.5 After removing unwanted points.

Frame-wise points selection The three areas of interest selected, represents the three levels of points to track. The point from the finger is the top line. The object square following the swipe will be in the middle and lastly, the reference points, which will be the lowest.

After these three areas have been decided, the algorithm iterates through the vector containing all the bounding rectangles for each frame. The bounding rectangles are sorted in three new vectors, one for each area of interest. These vectors also have one index per frame. If there are no bounding rectangles found in the respective frame and area, a rectangle with negative coordinates is stored instead. The vector that stores the reference points may store several points per frame. The other areas should only store one point per frame. If there are several points in these two areas found, the one closest to the swiping direction is stored.

When all the frames have been iterated through, the mean width of each area's rectangles is calculated.

There may occur issues where one rectangle is seen as many small rectangles instead. This is not an issue since it is the position and not the size that is important.

Calculate on reference dots When using the Epson robot, only the distance between the reference dots is of interest. This is done by iterating through all the stored reference dots and calculating the mean distance.

The 3D-printer also calculates the mean distance the same way. But it also need to add the movements of the reference dots. This is done by tracking the reference dots closest to the swiping direction.

When a new reference dot enters, the algorithm have to adjust. By taking the newly entered reference dot and comparing it to the second closes, the distance is removed as an offset.

It is important, when tracking the reference dots, to track the corner opposite to from where they enter the video. In the image below, if the left corner would be tracked, the reference dot would look stationary until it has entered the screen fully. By tracking the right side, the correct movement is tracked.

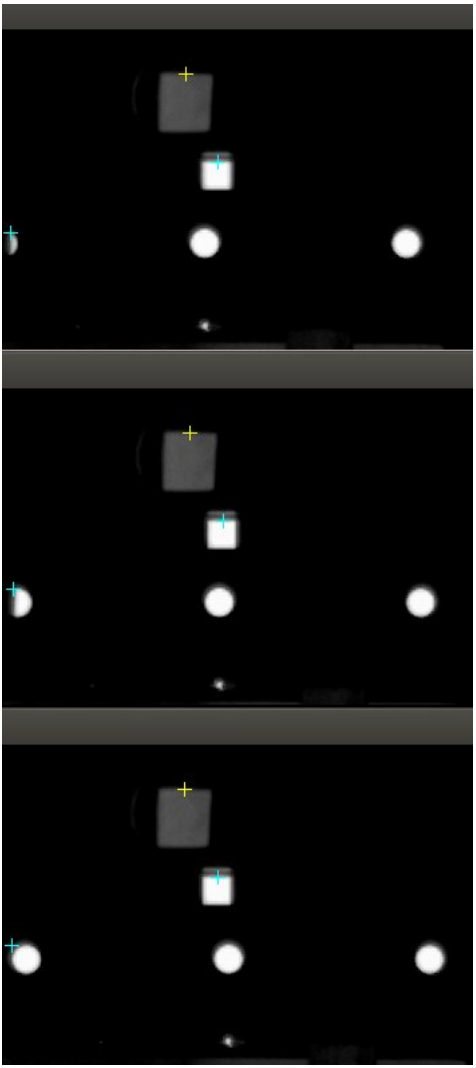


Figure 6.6 Looking at the bottom left dot. If the left corner is what is being tracked the following situation will make the leftmost reference dot appear stationary. When adding the width, the right corner will be tracked instead and the dot's movement can be appended correctly.

Debugging It is possible to use different debug modes when running the script. This is useful when checking for errors or to determine if a deflecting behavior is because of issues with the product.

Debugging mode 1 will allow the user to specify threshold levels for the edge

detection. It will also print out all tracked points and selected areas as shown in Fig 6.4 - 6.5.

Debugging mode 2 will also add a step by step iteration through the video file, printing out the tracked dots as seen in Fig 6.6.

Post processing

The tracked points are sent into the post-processing software together with the mean reference dot distance.

Transforming values The values sent in are rearranged so that they start from zero. By dividing the reference dots distance with the sent in actual distance in millimeter, a transformation constant from pixel to millimeter is retrieved. After these calculations, all the robot and object points are transformed from pixel position to millimeters and by using the time-vector created by the recording software, the times-steps are transformed from frame indexes to actual milliseconds.

Calculating results

Mean step-vector When all the values have been transformed to proper SI units the first step is to create a per frame step-vector, since there will be many measurements per frame. A 60 Hz display will have around 8 measurement points per frame when recording in 540 fps. It is practical to have a vector that takes only one mean measurement per frame step.

The theoretical distance between frame updates can be calculated to

$$\frac{vel_{robot}}{freq_{display}} \tag{6.2}$$

However this distance can vary quite a lot depending on frame-drops, noise, deceleration and acceleration or not syncing with the touch input (a touch display reads touch input in 90 Hz). A relax constant, a floating point number between 0-1, is introduced. This constants is multiplied with the theoretically calculated distance. The relaxation constant is determined by guessing and adjusted by empirical results throughout the project. The relax constant varies depending on swipe speed, where lower speed is harder to differentiate. It also has a lower value during the acceleration and deceleration phase. At the time of this report it has the following values:

```
relax = 0.6
if (distance_from_start <= 5 || distance_from_end <= 5)
{
        if (robot_speed > 100)
                relax = 0.2
        else if (robot_speed <= 60)
                relax = 0.4
        else
                relax = 0.3
```

```
}
else if (distance_from_start <= 10 || distance_from_end <= 10)
{
        if (robot_speed > 250)
                relax = 0.4
}
```

The theoretical time on each frame is equal to

$$\frac{1}{freq_{display}} \tag{6.3}$$

It should not be possible for the frames to update quicker, only slower, than the displays update time. But if measurements are missed there can be occasions where the updates can seem shorter, thus a relaxation constant is needed here as well. It is during the time of this report set to:

```
relax_time = 0.7
```

The position value of the step-vector is calculated from the mean value of all the position measurements in that step. The time value is taken from the last measurement.

Linear regression and standard deviation The created step vector is used to calculate the mean and standard deviation for step sizes of frame updates, both in time and distance. [Blom et al., 2010]

Linear-regression is used to get the line-equation for the robot and the object respectively. These line-equations give the velocities of the swipes which can be compared with the actual robot speed.

Relative distance Two vectors that plot the relative distances are created. One plots the relative time distance between robot and object. The other plots the relative length distance between robot and object over time.

Final output A .tex file is used as template to create a PDF with the results. The file is compiled with pdflatex and uses the package pgfplots for graph construction [*PGFPlots - A LaTeX package to create plots* Accessed: 2015-03-25].

The first plot will draw the absolute distance over time. This plots all the robot, object and step-vectors measurement points. It also plots the linear-regression models for the robot and the object. The second and third plot print the relative distances.

Below the plots, the measurement values are printed. Average time-step, calculated velocities, initial time for reaction and median time and distance offset.

Please refer to Appendix A for final results.

Warnings During calculations, results that appear odd in different ways are printed out as warnings. These are added to the end of the PDF. Currently the following warnings exists:

- 0 INIT LOW: If initial response time is very low compared to mean time delay. Usually a result of the post-processing having issues determining the first frame updated.
- 1 INIT HIGH: If initial response time is very high compared to mean delay. Could occur from the same reasons as short initial time or be an effect of frame drop or phone being slow at returning from inactivity.
- 2 VEL DIFF: If the calculated velocity differs a lot from the commanded.
- 3 REFDOT: If the computer vision was not able to distinguish the reference dots, no pixel to mm conversion is done.
- 4 TIME STEPS DIFF: A standard display has a 60 Hz refresh rate. This warning appears if the amount of found frame drops differs too much from the calculated amount. This can be an effect from frame drops or the algorithm having trouble finding the frame updates.
- 5 AVERAGE DISPLAY UPDATE LOW: If the average display update is lower than 60 Hz (16.6 ms). This should not happen so it usually means that the algorithm is having trouble.
- 6 AVERAGE DISPLAY UPDATE HIGH: If the average display update is higher than 60 Hz (16.6 ms). This can mean that the phone suffer from frame drops.
- 7 NOT FROM STATIONARY: If the recording starts in the middle of the swipe.
- 8 WARNING FRAME DROP: If a frame is more than twice as long as it should (60Hz) it is considered as a frame drop. It is important to know that the touch input is sampled in 90 Hz and is not synced with the display. So this means that an "unlucky" touch event could actually take 16.6 ms + 11.1 ms = 27.7 ms, in other words almost twice the display update.
- 9 WARNING STATIONARY ERROR: If the tracking object passes or stops before the finger, and if this distance is large, this warning appears.

Some of these warnings display additional data as well.

6.5 The Class structure

This section explains how the previous chapters are linked together for a final test system. This section can be seen as optional, and is meant for those interested in understanding the implementation more deeply.

There are three runnable programs in total. `./configure_robot`, `./coyote_run` and `./coyote_cv`. The `configure_robot` is used to add a new or edit an existing product, and it also adds and creates tests related to these products. `The coyote_cv program` performs the computer vision and plots the results. The `coyote_run` combines all the classes and runs the full automated test.

The structure

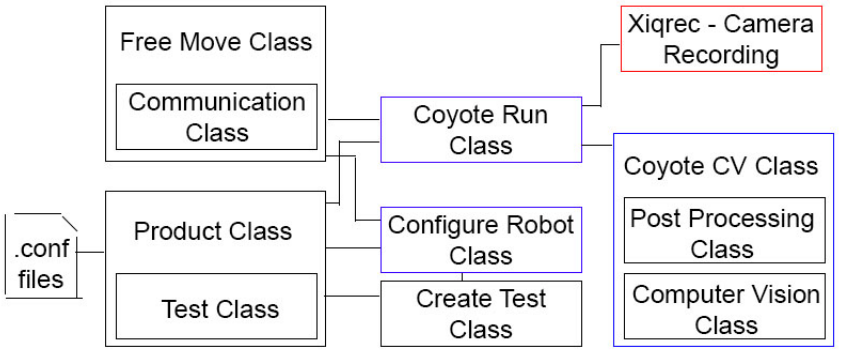


Figure 6.7 The class structure. Red border is standalone program. Blue borders are runnable classes.

Fig. 6.7 shows the architecture where red border is standalone program and blue borders are runnable classes.

`Robot Communication` and `Robot Free Move` have been explained in Chapter 6.2 (Robot) . The communication class takes care of the direct communications to the Arduino card while Free Move is the overlaying class which adds constraints and allows for running and creating tests.

`Coyote cv` and `Post processing` is explained in Chapter 6.4 (Computer Vision).

Xiqrec is the recording program explained in Chapter 6.3 (Camera).

Configure product

This is used to add new or edit existing products. The program uses Robot Free Move to create new hard coded tests, as explained in Chapter 6.2 (Robot). Configure product is used to store information on products and corresponding test cases. Whenever a product is created or updated it is stored in a .conf text file so that it can be loaded at any time.

When a .conf file is loaded the information is stored in a product class. The class contains all the needed information of the product as well as Test classes. The test

classes are basically containers containing init, main and end list. Each being a list of strings with G-code commands.

Running Configure Prouduct provides the following options: 1. Add product, 2. Edit existing product 3. See list of existing product, 4. Play with the robot or 0. Exit.

Add product

Adding a product will first ask for the name of the product and then step by step ask for the physical information to be added. Everything is stored in a .conf file.

Edit existing product

When this option is selected the user will select a .conf file. Whenever the user chooses to save and exit the old .conf file is over-written. Running Edit exiting products gives the following options: 1. Change constants, 2. Delete test, 3. Add test, 4. Run tests, 5. Delete product 9. Save and exit or 0. Exit without saving.

Tests

Every product contains a list of tests associated to that product. The class contains information about the test: name, type and if a camera is needed. It also contains three vectors init, main and end. The init vector sets up the test. The main vector performs the test itself and should be loopable. The end vector restores the changes made during the test.

Create Test Class

Many tests can be calculated automatically by knowing the product's measurement. These are generated for the product every time it is loaded and have a separate category from the hard-coded tests.

The `Create Test Class` generates the automated tests.

The .conf file

The products are stored in a .conf file. The file has the same name as the product, always using large capitulation. The first part of the file defines all the measurements and constants needed.

```
#DEFINE CONST
NAME = LUMIA
HEIGHT = 80
WIDTH = 50
RESY = 800
RESX = 480
DEPTH = 10
OFFSETY = 20
OFFSETX = 7
WEIGHT = 141
FREQ = 60
#END CONST
```

The second part defines the tests.

```
#DEFINE CTEST
CT1
CT2
CT3
#END CTEST
#DEFINE TEST
T1
T2
T3
#END TEST
```

Under the #GCODE tag the code for the hard-coded tests are provided. "(NAME" gives the start and ")NAME" gives the end. The different vectors are separated with HEAD, BODY and BOTTOM. A person who knows G-code would probably add or edit custom tests directly in the text-file which can be tested in the `Configure Robot` program.

```
#GCODE

(T1
HEAD
BODY
M204 S3000
G01 F6000
...
G01 Z10
BOTTOM
)T1
```

6.6 Automating the tests

The `Coyote Run` class is the actual automation class. Inputs to this program are the product name, which test to be performed, a vector with velocities and the distance between the reference dots on the phone app.

A simple pseudo code for what the program does is as follows. Below is an in-depth explanation of the steps.:

```
load_product_values_and_tests

start_robot
execute_test_initiation
create thread1, thread2

move_robot_to_init_position
loop
{
   thread1 : start_camera_rec
   block_read fifo
   thread2 : execute_test_main
```

```
    wait_for_threads_to_finish
    run_computer_vision
    store_resulting_pdf
    encode_video   //optional
    save_video_to_hard_drive   //optional
}
        end_test
```

Initiation

The loading of values is performed through the `Configure Product` class. It opens the respective product's .conf file and loads the values explained in the test section above in this chapter. Through the same .conf file the tests belonging to the respective product are loaded.

During the initiation phase output folders are created as well. Every test result will end up in the following output.

archive/DATE/PRODUCTNAME/PRODUCTNAME-DATE-TIME-VELOCITY.pdf

Right before the loop the robot starts the initial phase of the test. All the tests are run through the `Robot Free Move` class.

The loop

The camera recording software is started by using a `system()`-call in a separate thread. The Xiqrec program will take some time to initialize and write to FIFO when it is ready to start recording. The main program will block until FIFO is read, after which, the main part of the test starts in another separate thread. The time of the recording is adapted depending on the speed of the swipe and size of the display. The threads are then synchronized using `join()` so that the program will block until both the camera recording and the robot's movements have finished.

In the next step, the computer vision software is performed on the newly recorded video file. The video file is still located in the RAM-memory and in raw-format. When analysis of the video has finished and the post-processing have stored the results to pdf, the video file is encoded and archived to x264 format to preserve space.

Lastly the encoded video file, the pdf, and the time-vector from the recording software is moved to the archive folder, sub-foldered by date and product name.

This loop is performed once for each index in the sent in velocity vector.

7

Experiments

This chapter contains all tests that have been performed during the project. The tests have all been performed during different stages of this thesis and the order has nothing to do with when the tests were done. Reading through the report will give better understanding of the reason for these tests.

7.1 Identification of the robot

Acceleration Test using different voltage and weights

This test will be to determining the printer's maximum acceleration and how the reference voltage over the micro-controllers effects the possibility of increasing the acceleration. A strong acceleration will waste less time on acceleration and deceleration and allow for greater velocities.

The X-axis is heavier loaded and will probably not be able to reach as high acceleration as the Y-axis (the X-motor carries the Y-motor). On the other hand, the Y-axis is probably more load sensitive because of that. The currently heaviest product is the Z2 Tablet at 495 grams.

The test result provides the maximum acceleration achieved with different weights. It was performed on the original heating bed plate, before the modified fixture plate had been created (seen in Fig. 5.5).

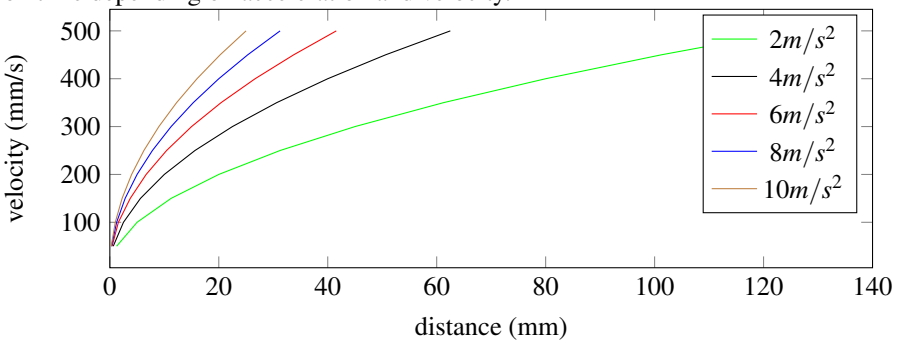
According to the manual the recommended reference voltage on the micro-controllers is 425mV. At the support forum there have been cases where users have been advised to increase the reference voltage to 550mV, however going above that is not recommended without external cooling. During this test the acceleration in X, and Y was limited to 10,000 mm/s^2 in the firmware.

Sony demands to be able to run swipes in 400mm/s.

Acceleration test of Vellman K8200 (mm/s^2)						
	X-axis			Y-axis		
Weight	425mV	550mV	600mV	425mV	550mV	600mV
0g	4300	5500	6000	4900	10,000(MAX)	10,000(MAX)
100g	3900	5100	5800	5400	8500	10,000(MAX)
200g	3800	5000	5800	5000	9100	10,000(MAX)
300g	3700	4800	5600	4400	8500	9600
400g	3500	4500	5500	4400	6700	7800
500g	3000	4500	5200	4100	6100	7600

Table 7.1 Max acceleration of K8200 tested with different reference voltage and weights.

To understand the result, the following will visualize the acceleration and deceleration time depending on acceleration and velocity.



This has been visualized by the following calculations.

$$\begin{aligned} d &= \int_t \int_t a dt = \frac{at^2}{2} + v_0 t + d_0 \\ v &= \int_t a dt = at + v_0 \end{aligned} \tag{7.1}$$

Since motions are relative and starts from zero velocity this can be rewritten.

$$\begin{aligned} d &= \int_t \int_t a dt = \frac{at^2}{2} \\ v &= \int_t a dt = at \Rightarrow v^2 = (at)^2 \end{aligned} \tag{7.2}$$

$$\Rightarrow d = \frac{v^2}{2a}$$

Including both acceleration and deceleration, assuming both are the same, doubles the distance.

$$d = \frac{v^2}{a} \tag{7.3}$$

Acceleration test using final mechanical modifications.

This is performed on the final modified 3D-printer with the correct mounting possibilities and on actual products. The purpose is to append a simple model for the robots achievable acceleration.

This test will perform swipes in X- and Y-direction as well as diagonal. Each test is performed 100 times and it will only pass if there are no malfunctions on the stepper motors. Every X- and Y-test is a back-and-forth swipe so 100 passes accounts for 200 swipes. The diagonal test goes from corner to corner in the order (x,y): (0,0),(1,1),(1,0),(0,1) so that 100 passes accounts for 400 swipes where half of them are diagonal swipes.

Each test is performed three times with rest in between.

Acceleration test of Vellman K8200 (mm/s^2)									
500mm/s									
Weight	Y-axis			X-axis			Diagonal		
Empty(0g)	11000	10000	10000	5000	5000	5000	FAIL	FAIL	FAIL
Z1 Compact(137g)	8000	9000	9000	5000	4500	5000	FAIL	FAIL	FAIL
Nexus 7(340g)	7500	8000	8000	4000	4000	4000	FAIL	FAIL	FAIL
Z2 Tablet(444g)	7000	7000	7500	3500	3500	3500	FAIL	FAIL	FAIL
450mm/s									
Weight	Y-axis			X-axis			Diagonal		
Empty(0g)	11000	11000	11000	5000	5000	5500	FAIL	FAIL	FAIL
Z1 Compact(137g)	9000	10000	9000	4500	5000	4500	FAIL	FAIL	FAIL
Nexus 7(340g)	8000	8000	8000	4000	4500	4500	FAIL	FAIL	FAIL
Z2 Tablet(444g)	7000	7000	7500	4000	4000	4000	FAIL	FAIL	FAIL
400mm/s									
Weight	Y-axis			X-axis			Diagonal		
Empty(0g)	10000	10000	11000	5500	5000	5500	7500	7500	7500
Z1 Compact(137g)	9000	9000	9000	4500	4500	4500	6500	7500	7500
Nexus 7(340g)	7500	7500	7500	4000	4000	4500	6500	6500	7000
Z2 Tablet(444g)	7000	7000	7000	4000	4000	4000	6000	6000	6000

Table 7.2 Max acceleration of K8200 tested with different velocities.

7.2 Resolution test

The specifications of the product can not be changed. The camera has VGA resolution and 540 frames per second. Slower swipes require more from the resolution because there will be smaller differences between frames. The following is studied with the lowest required swipe speed, 40 mm/s, to get a worst case scenario.

Limitations from camera - Epson robot Considering the Epson model, the resolution will differ depending on the size of the product. The camera needs to cover the entire display since the product will be stationary. A 10" tablet display will be 220

mm wide. Assuming that 80 % of the 640 pixels are used efficiently, the distance per pixel covered will be:

$$\frac{220mm}{640pixels \cdot 0.8} = 0.430mm/pixel \tag{7.4}$$

A slow swipe of 40 mm/s, will then result in

$$\frac{0.04mm/ms}{0.430mm/pixel} = 0.093pixel/ms \tag{7.5}$$

In other words 10.75 ms/pixel. The 540 fps will average at 1.85 ms per frame, assuming the recording speed is constant. This means that 0.172 pixel/time-step or slightly less than every 6th frame will actually track a movement. It also means that a 60 Hz display, that updates every 16.7 ms will move:

$$\frac{1000}{60s^{-1} \cdot 10.75ms/pixel} = 1.55pixels \tag{7.6}$$

Limitation from camera - 3D-printer Since the 3D-printer moves the product, the camera is stationary while different areas of the product passes by its focus area. This way, the camera can be in the same position no matter what size of the product.

To calculate how much space is at least needed, the fastest swipe and a high delay should be considered. The fastest swipe measured will be 400 mm/s, assuming that at least delays up to 200 ms should be measurable gives

$$400mm/s \cdot 0.2s = 60mm \tag{7.7}$$

This means, at least 60mm of the display is needed in view, otherwise the object dot will move out of view. In the calculations some margin is added.

$$\frac{60mm}{640pixels \cdot 0.8} = 0.117mm/pixel \tag{7.8}$$

A slow swipe of 40 mm/s, or 0.04 mm/ms, will then result in

$$\frac{0.04mm/ms}{0.117mm/pixel} = 0.342pixel/ms \tag{7.9}$$

In other words 2.93 ms/pixel for 60 mm range. Assuming a constant 1.85 ms per frame time sampling, same as with the Epson robot, means 0.633 pixel/time-step. This means that about every 2nd frame will actually track a movement. It also means that a 60 Hz display, that updates every 16.7 ms, will move:

$$\frac{1000}{60s^{-1} \cdot 2.92ms/pixel} = 5.707pixels/frame \tag{7.10}$$

Limitations from product display The resolution of the display might also be a limiting factor when swiping in lower velocities. The interesting information to look at in this case is the pixels per millimeter ratio of the display. This is usually worse for budget phones or large tablet. Sonys current budget model in Sweden is the Xperia M2 which has a 4.8" display with 960x540 resolution. The gives the display a length of 106 mm and 0.116 mm/pixel density. A 40 mm/s swipe will then move:

$$\frac{0.04mm/ms}{0.116mm/pixel} = 0.345pixel/ms \tag{7.11}$$

This could be re-written as 2.9 ms/pixel.

Testing the resolution Ten 40 mm/s swipes are tested both when the recording width is 140 mm and 80 mm. The warnings represent the warnings explained in the post-processing section of Chapter 6.4 (Computer Vision).

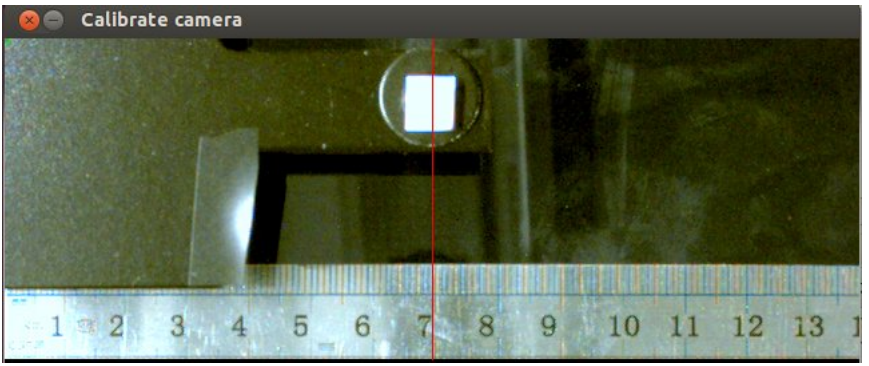


Figure 7.1 140 mm width.

Resolution test 40 mm/s swipes				
Test	140 mm		80 mm	
#	Warnings	Results	Warnings	Results
1	6	Update: 17.69±4.00 ms Delay: 139.67 ms		Update: 17.07 ±2.41 ms Delay: 145.56 ms
2	6	Update: 17.84±4.61 ms Delay: 138.00 ms		Update: 16.91±2.09 ms Delay: 146.73 ms
3	0: 44.8% 6	Update: 18.08±5.70 ms Delay: 145.11 ms		Update: 17.13±2.83 ms Delay: 151.13 ms
4	0: 2.2% 6	Update: 17.83±3.90 ms Delay: 139.75 ms		Update: 16.91±2.47 ms Delay: 144.89 ms
5	0: -212.0% 6	Update: 18.56±6.89 ms Delay: 139.50 ms		Update: 17.22±3.05 ms Delay: 151.33 ms
6	0: 5.5% 4:-5.6 % 6	Update 17.96±4.23 ms Delay: 132.80 ms		Update: 16.83±2.01 ms Delay: 155.00 ms
7	6	Update: 17.66±3.91 ms Delay: 143.22 ms	0: 68.2 %	Update: 16.86±2.10 ms Delay: 148.44 ms
8	0: 42.4%	Update: 17.36±3.31 ms Delay: 139.18 ms		Update: 16.75±1.49 ms Delay: 147.67 ms
9	4: -7.1% 6	Update: 18.24±4.79 ms Delay: 139.11 ms		Update: 16.90±2.22 ms Delay: 145.5 ms
10	4: -5.0% 6	Update: 17.73±3.70 ms Delay: 142.8 ms		Update: 16.82±1.94 ms Delay: 145.5 ms

Table 7.3 Comparing results from 140 mm and 80 mm recording distance. The closer alternative generate a lot less warnings and smaller standard deviations.

7.3 Robustness and quality

When the final automated lab setup is complete, multiple tests can be run to get an understanding of the robustness.

To investigate how sensitive the software is to bad calibration a quick angle test is performed. The swipe test is performed when the camera has been positioned with a steep angle.

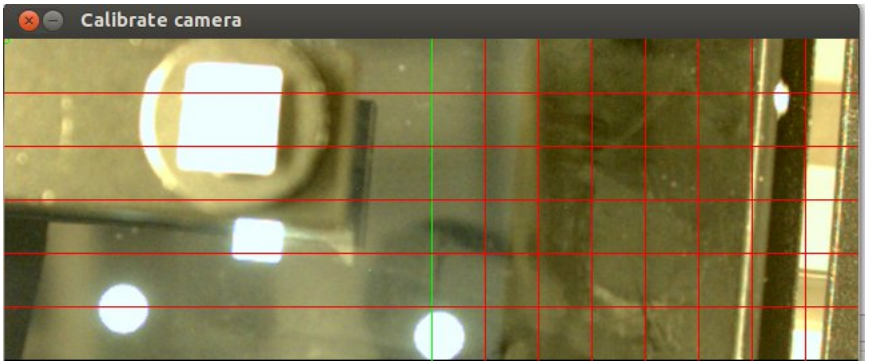


Figure 7.2 A steeper angle then this is still possible but gives a lot less measurement points.

The algorithm still manages to manage pretty steep angle. This is of course not a recommended setup since the results will be incorrect because of trigonometrical differences. The robot's X- and Y-directions should be parallel with the camera for best result.

Final testing

The final testing consists of three parts and all the data can be found in Appendix. The first part shows swipes for two different products on both the Epson and the 3D-printer. It provides the full result file. These tests have been done hundreds of times and the result files in Appendix A only reports a representative handful of these.

The second part compares the Epson and 3D-printer relative delay for each velocity.

The third part plots 10 swipes at the same velocity, product and robot. This it to show how much the delays actually differ from run to run.

7.4 Cost comparison

The following table shows a price estimate of the two systems. It could also be worth to mention that the amount of space that the Epson robot takes with its cage is about $4m^3$ while the K8200 with cages takes about 1 m^3.

Cost comparison SEK		
	EPSON	K8200 3D-printer
Robot:	150,000	5,500
Fixture:	20,000	4,000
Cage:	30,000	1,000
Computer:	4,500	4,500
Camera:	9,000	9,000
TOTAL:	214,500 + installation cost	24,000 + building time cost

8

Discussion

Final thoughts on tracking software

Currently there is not much done in the software to avoid stationary noise. When using the algorithm in debug mode it is possible to crop the video to remove noise from the borders, but this is not a good option in the fully automatic case. Using the Epson robot requires a lot of manual steps so it uses some extra debugging to remove stationary noise. On the 3D printer these steps are not possible but because of the 3D-printer's way of moving the sample the noise situation should be opposite to the Epson robot. The notification LED on the phone would be stationary on the Epson and not so on the 3D printer. On the other hand reflections from the robot would be stationary on the 3D-printer while moving for the Epson. The printer is therefor set up in a dark environment too avoid noise such as outside light or reflections.

Adding a lot of noise filtering may add a lot of special cases which makes the algorithm more dependent on specific conditions. Before adding more noise reduction an optimal environment for recording should be set up and noise reduction can be added when needed. Currently the tests perform well in an open lab setup and will of course perform even better when inside an isolated area. The whole idea is to measure delays and to catch errors on the products, adding to much noise filtering may end up filtering real measurements.

Discussing the resolution

This summarizes the resolution calculations made in Chapter 7 (Experiments).

Calculated result from the Epson 1.55 pixels per display update is not a very high resolution. It is a realistic assumption that the tracking software will error with at least one pixel. Consider e.g., a tracked dot in between pixel A and pixel B. It will be very hard to differentiate the frame updates in this scenario and it also renders the time-step resolution unnecessarily high. In other words, it would be better to have lower recording speed and higher resolution for this scenario.

In this setup, to get a reasonable resolution which will be able to differentiate the frame updates, either the swipes will have to be faster, which will provide a better pixel per ms ratio, or the camera will have to focus on a smaller area of the display.

The first case would mean not reaching the requirements made by Sony Mobile and the second would result in less measurements.

Calculated results from 3D-printer 5.7 pixels will be a lot more easy to differentiate. The 0.684 pixel/time-step means that the time versus display resolution is a lot more reasonable in this setup.

Test results In Table 7.3 a lot more warnings are given when recording from a greater distance. The most common warning, code 0, means that the initiation time calculated is relatively short compared to the median time delay. This indicates that the program is having problems differentiating noise from when a move has started. Code 4 means that there are more than 5% difference in the amount of frame updates found and what would be the theoretical amount. Warning 6 means that the mean-time for each frame update is more than 5% larger than the theoretical. Basically all of these warnings indicate that the algorithm is struggling to differentiate the movements. Studying the Results column, it is clear that when the distance is 140 mm it has a larger standard deviation.

However, considering that there is only around 2.85 pixel difference between frame updates in the more distant case, it still provides surprisingly good values. Also note that the most distant of these two cases covers less then the size of a 10" tablet and the values would be even worse in that situation.

Why 3D-printer gives more reliable results

The reason for why the 3D-printer gives more reliable results is because of how the camera and arm is kept stationary while the product is moving. This way, the swiping finger will always be at the same position. The tracking object, which is the object on the display set to follow the finger, should be in the same position at start and at the end of the swipe. This means that offsets can easily be detected. During the swipe this object will have a delay from the finger and that delay will be around the same area from the cameras perspective throughout the swipe. This means that distortions will have more of a constant effect since the finger and tracking dot will be situated in the same area during the swipe. As an example consider a constant time delay, where it would have an object point at a fixed position from the swiping finger during constant speed.

In contrast, the Epson robot instead moves from one side to the other. That way it will be harder to know the effect of distortion from lens, perspective or a bad setup of the camera.

9

Conclusion

Acceleration tests

Looking at Table 7.1 it is clear that a higher voltage provides more power. But without any extra cooling the reference voltage should be kept to 550mV at maximum, as was recommended by the developers at the support forum.

Looking at Plot 7.1 an acceleration of at least 6000 mm/s^2 should be desirable. It is clear that swipes should be done in the Y-direction when comparing this with the tests.

The second test shows that there is an issue with higher velocities. Running above 400 mm/s fails the diagonal test no matter what acceleration. When doing a hundred diagonal tests usually 4-8 of them fail, even when running at low accelerations. Since there is no requirement to go beyond 400mm/s, this is not investigated any further.

Following the results of Table 7.2 with the new fixture for mounting products. This allowed for using actual products when testing and also a more robust fitting of the sample. The results where also more stable than in Table 7.1. This makes it more suitable to estimate maximum acceleration and weight dependencies.

By using the weight of the product as an input parameter, a simple model for giving the maximum acceleration in each direction is created from the values in Table 7.2. The following simple linear model is used.

$$\begin{aligned} a_y &= 10000 - weight \cdot \alpha \\ a_x &= 5000 - weight \cdot \beta \end{aligned} \tag{9.1}$$

Using the values in Table 7.2 the values was set to $\alpha = 10$ and $\beta = 1.2$. The diagonal moves are performed in a 45 degree angle, which means that both motors will contribute equally. The maximum acceleration should theoretically be decided by the weakest motor hence

$$a_{diag} = \sqrt{2} \cdot a_{x/y} \tag{9.2}$$

Looking at the results of the diagonal tests in Table 7.2, the acceleration achieved at 400 mm/s seem to correlate quite well with Eq. 9.2.

To evaluate the model a simple test was performed going back and forth from origin to end point. Starting from bottom left corner (origin) to top left corner and ending from origin to bottom right corner gradually increasing the angle. This test was done multiple times with gradually increasing masses.

Resolution

Clearly, using the Epson robot, the disadvantages of the camera position will have to be considered during slower swipes and larger displays. Only swiping on a part of the display is an option but would give less measurements and would above all not test the entire display.

The 3D-printer can provide better camera resolution without having to sacrifice the length of the swipes. It will also give a consistent resolution independent on the size of the product.

To conclude this section, there definitely can be losses because of resolution when using the Epson robot. For the 3D printer, the equipment used seems to have a good balance between time and video resolution.

Comparing results between robots, Appendix

The overall time delay seem about the same overall. Higher velocities seem to have larger differences but also do not seem to favor any of the robot. There are some initial differences between the two robots on the tablet measurements, where the 3D-printer show, more delay initially. Also the 3D-printer seem to show, when swiping on the phone, a larger delay in the end for higher velocities. These differences will not be investigated further throughout this report, it is not the purpose of this thesis, but it is worth noting the anomalies.

One important measurement is the calculated robot velocity, seen in the respective results of Appendix A.1. It is when the OpenCV uses linear regression to calculate the velocity of the swipe. This can be compared with the velocity reference. An interesting aspect is that, in every single swipe, the 3D-printer produces a very accurate calculated result, which is a lot better than the Epson robot. Except for the highest velocity it differs with less than 1 mm/s. This indicates that the measurements from the 3D-printer are more accurate. This is most likely due to the resolution and positioning issues that have been discussed. It does show that it is more important to get an as close camera position as possible, rather than having the best robot. Also the 3D-printer does perform impressively accurate for its low cost.

Comparing result on the same robot

When comparing the results between different products and robots, we could see quite large time-delay differences. Especially for higher velocities. In Appendix A.3 10 swipes are made with the 3D-printer, on the same product, using the same velocity. This is done for 40, 100 and 333 mm/s.

There seems to be a variance in the time-delay which becomes larger with increasing velocities, much like what was seen when comparing Appendix A.1 and AppendixA.2. Looking at the results it also seems as if the delay stays rather constant. In other words, a swipe that starts off with a high delay will maintain a high delay throughout the sequence. A speculative reason for this could be that a touch display samples touch-events at 90 Hz and the displays updates the graphics in 60 Hz. Depending how these are synchronized, it could cause a difference in the delay of the results.

What also can be establish by the measurements of Appendix A.3 is that the trends seem to be very similar throughout the move. This implicates that the behavior of the robot and the touch device is quite consistent.

9.1 Final setup

Clearly the 3D-printer manages to swipe from 40-400 mm/s which was the requirements from Sony. We could also see that the trajectories from the 3D-printer gave better velocity results than on the Epson robot. Comparing the results of the robots the results where different but still close enough considering the differences given by A.3. It was possible to see some different behaviors on especially the tablet between the two robots. To fully evaluate the reason for these, more test would have to been done.

The automated test-system using the 3D-printer works very well for measuring swipes. It even gives better results than the Epson, which states that it is important to consider all the aspects before setting up a new system. The benefits of using the Epson robot would be useless unless better camera positioning is set up. It will be possible to set up 9 times as many 3D-printer systems than Epson system for the same price. These will also not cover nearly as much volume.

A lot of other test could of course also be implemented, in many cases UI tests do not even need an advanced camera, and a simple normal frame rate camera might suffice. This would drop the price of the final lab setup with even more.

10

Epilogue

During the months that have past since the finish of this project, some new improvements have been made. The final setup has been integrated with a server-based system which will allow new updates and test jobs to be triggered automatically as well as re-flashing of the phone. The final lab setup has been improved where the 3D-printer has been installed inside a locker to keep out light, this will help eliminating noise from the outside. The system has been wired together with a relay so that the USB connection can be triggered on and off. Also a LED have been installed on the robot finger so that no external light is now needed for tracking. A future goal is to have the LED turn on and off on touch so that taps can be measured. A few new test cases are being integrated on the system as well.

It has been decided to order more of these 3D-printers so testing can be expanded even further and plans have been made to integrate the control software with a standardized robot control system at Sony Mobile.

A

Swipe Measurement Results

The products on which the tests have been performed are not mentioned in this report. Tested products can be of any brand, and have often been old prototypes, running on both hardware and software that never even reached final release. Please note that the results are for comparing and supporting this project and should not be seen as actual consumer product results.

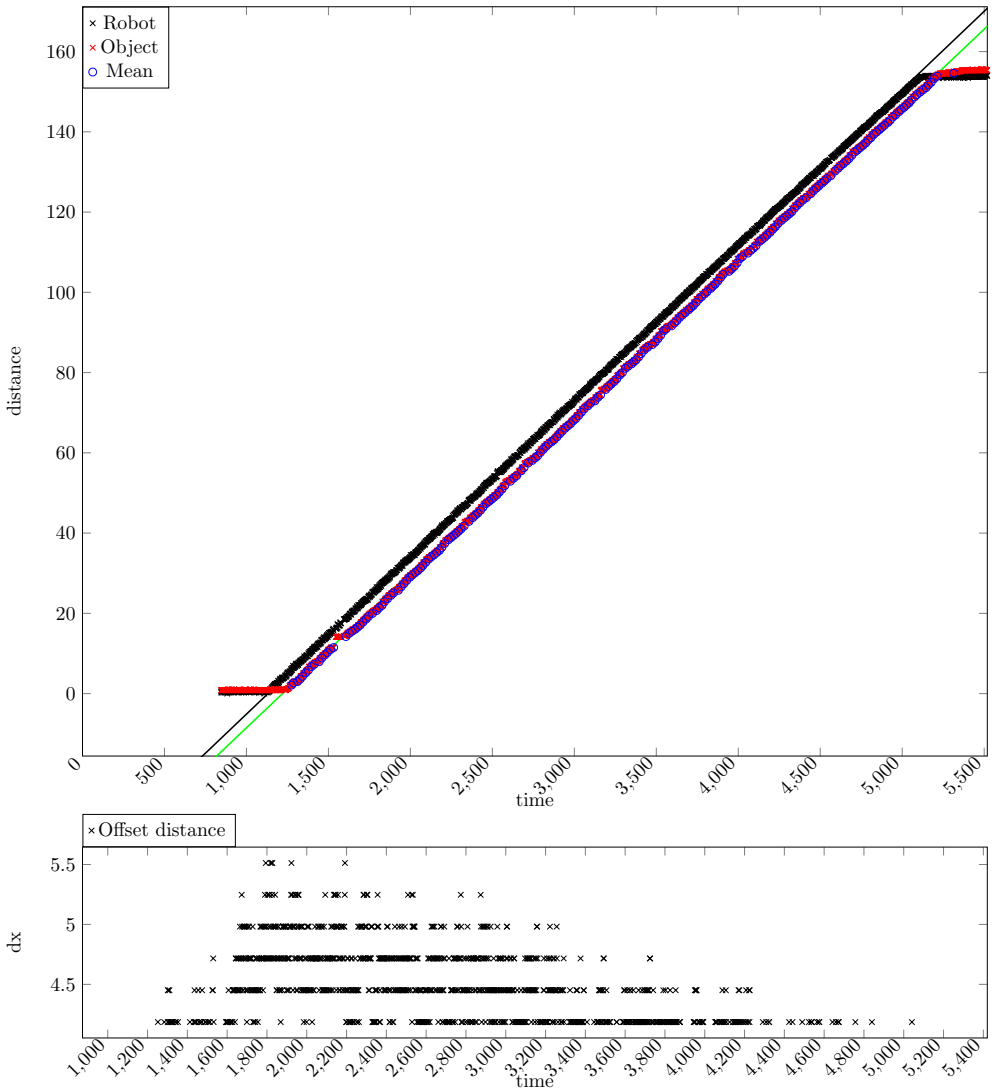
A.1 Result files 40-400mm/s on both robots

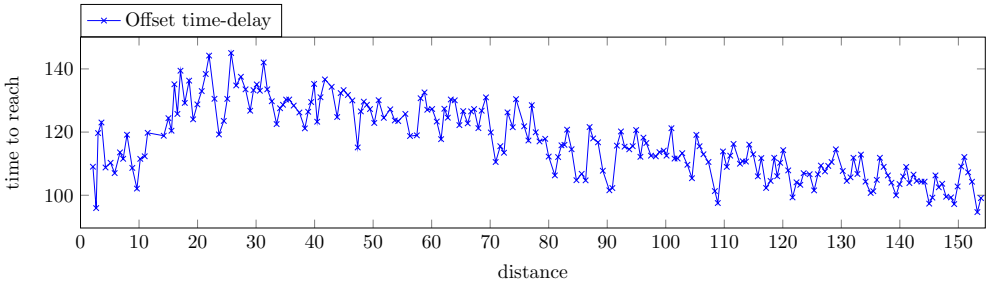
The following result files are the final pdfs that are created for each swipe in a test suite.

Swipe results Tablet 40-400 mm/s on Epson

1 Tablet 40mm/s Epson

1.1 Empirical Result





1.2 Data

Average display update time: 18.1488±5.66394ms
Average display update step: 0.701662±0.215751mm
Robot actual speed: 40mm/s
Calculated average robot speed: 38.9235±0.373025mm/s
Calculated average object speed: 38.6907±0.527485mm/s
The robot starts moving at time 1145.07 and it takes 130.933 ms for the screen to respond
Estimated touch of display: 764 ms. Estimated time for screen response: 81ms (quite uncertain method estimate ±10 ms)
Median time delay: 115.545
Median distance offset: 4.4508

1.3 Warnings

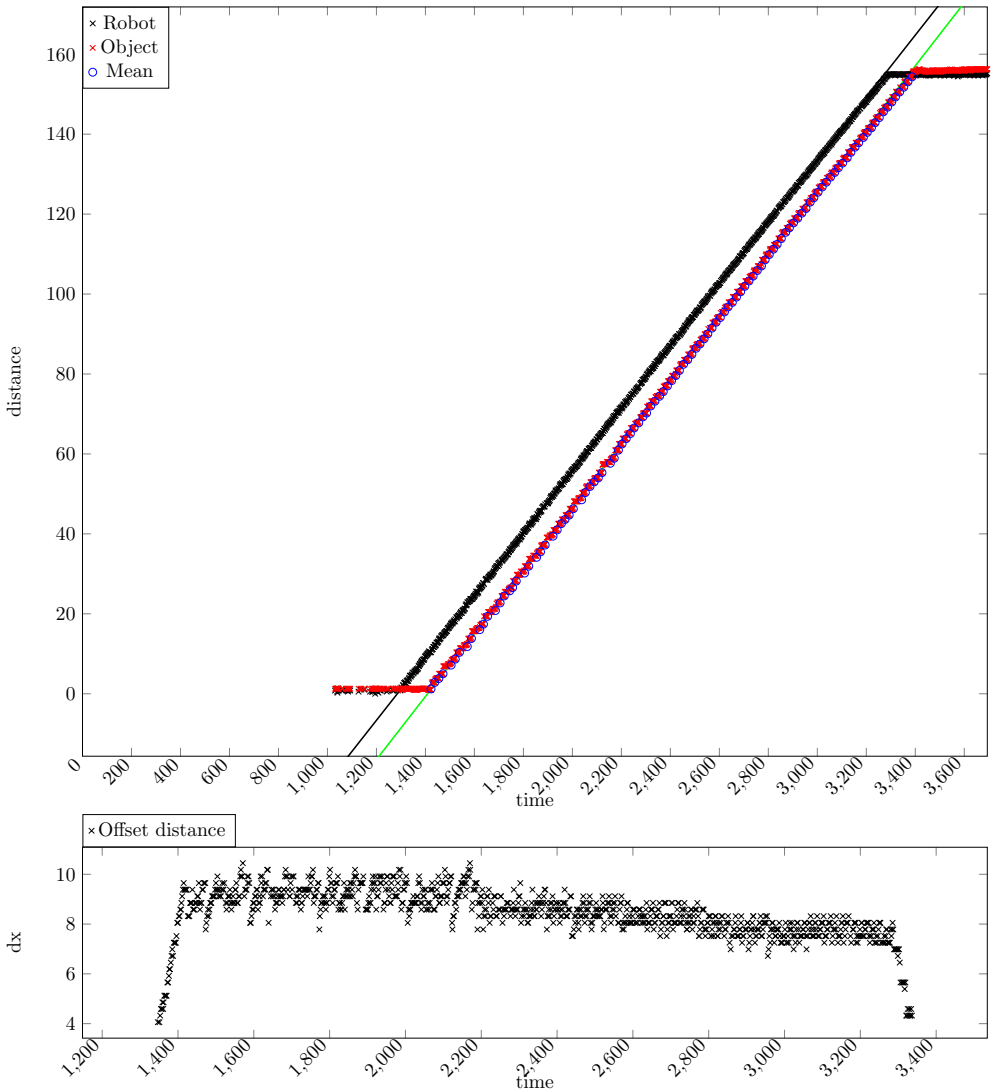
- WARNING 8: Suscpected framedrop(s) detected.

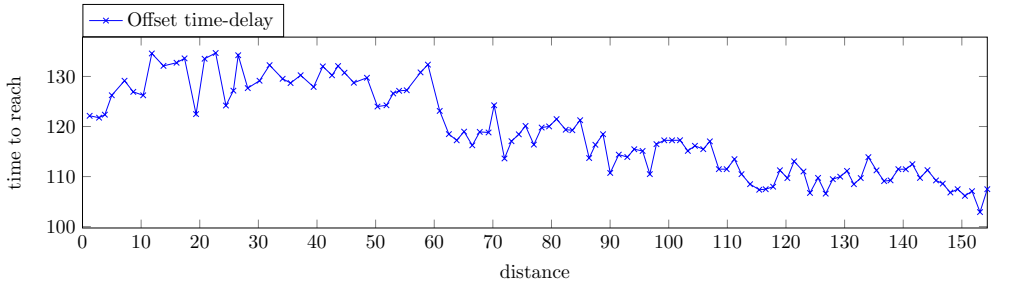
1 Tablet 80mm/s Epson

1.1 Information

...

1.2 Empirical Result





1.3 Data

Average display update time: 18.066±4.71703ms
Average display update step: 1.40839±0.30313mm
Robot actual speed: 80mm/s
Calculated average robot speed: 77.9138±0.328818mm/s
Calculated average object speed: 78.8928±0.525358mm/s
The robot starts moving at time 1295.5 and it takes 127.503 ms for the screen to respond
Estimated touch of display: 909 ms. Estimated time for screen response: 122ms (quite uncertain method estimate ±10 ms)
Median time delay: 117.24
Median distance offset: 8.58284

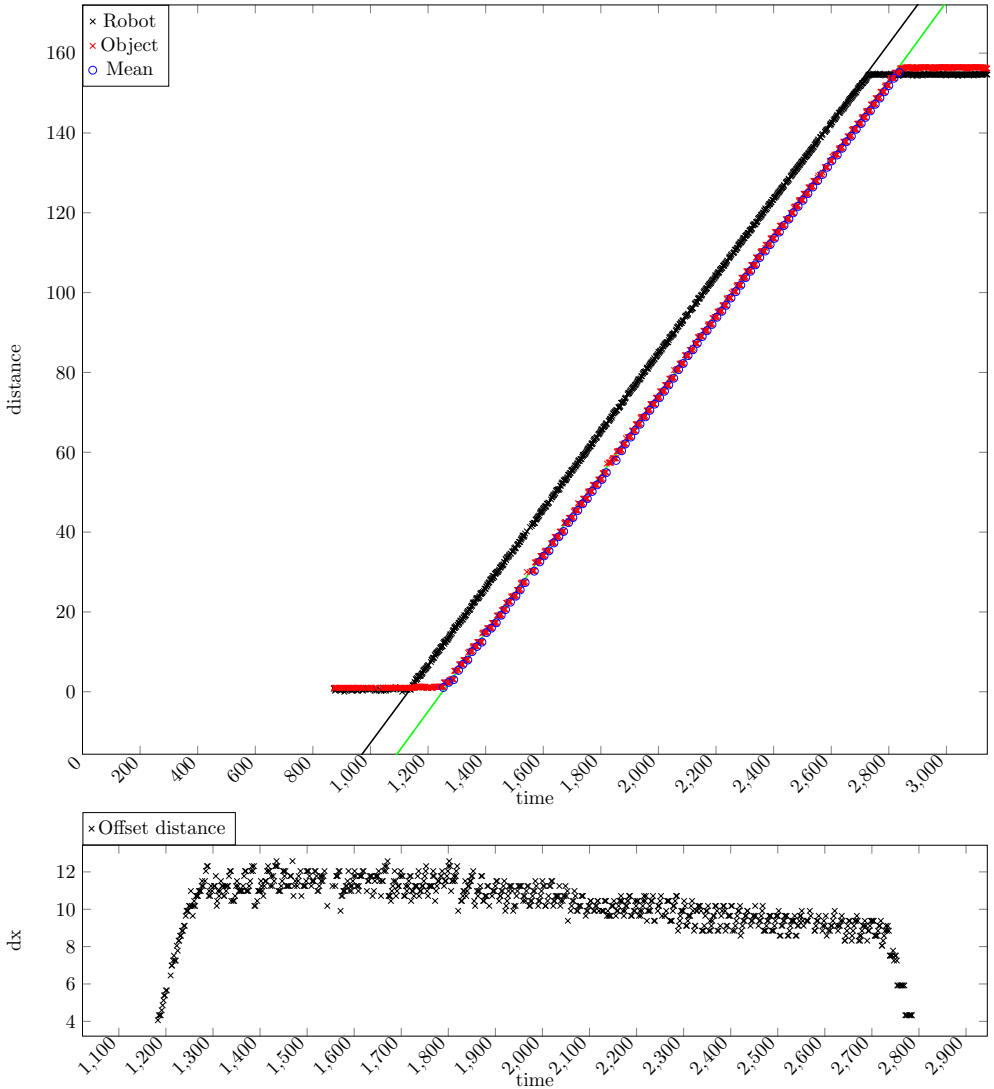
1.4 Warnings

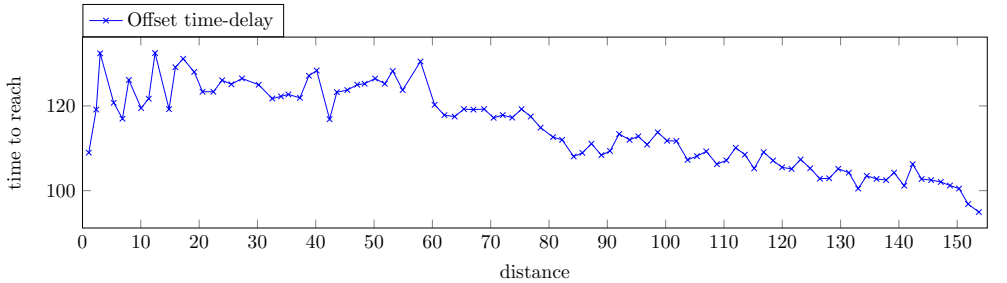
1 Tablet 100mm/s Epson

1.1 Information

...

1.2 Empirical Result





1.3 Data

Average display update time: 16.9889±2.66873ms
Average display update step: 1.6743±0.320941mm
Robot actual speed: 100mm/s
Calculated average robot speed: 97.3272±0.306738mm/s
Calculated average object speed: 98.7598±0.553709mm/s
The robot starts moving at time 1139.78 and it takes 113.224 ms for the screen to respond
Estimated touch of display: 743 ms. Estimated time for screen response: 130ms (quite uncertain method estimate ±10 ms)
Median time delay: 113.777
Median distance offset: 10.4456

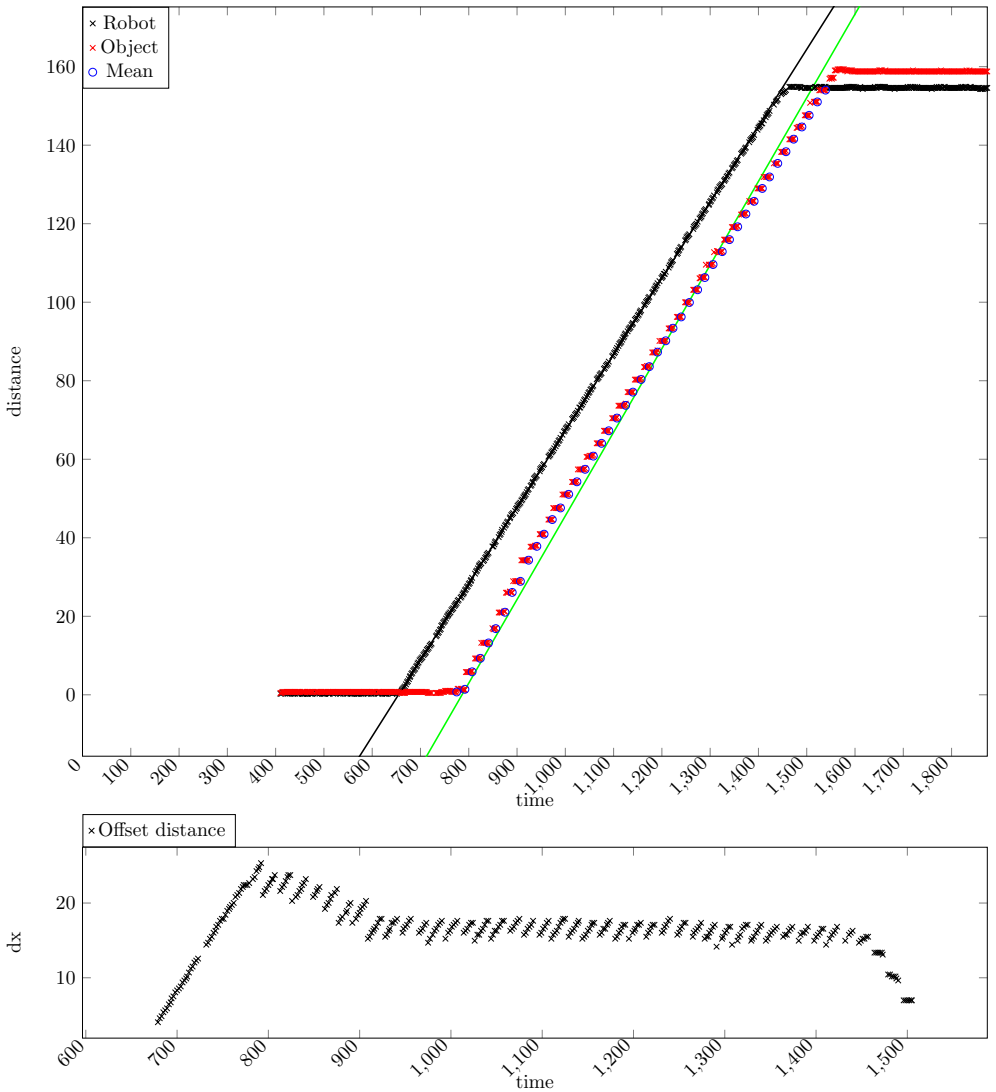
1.4 Warnings

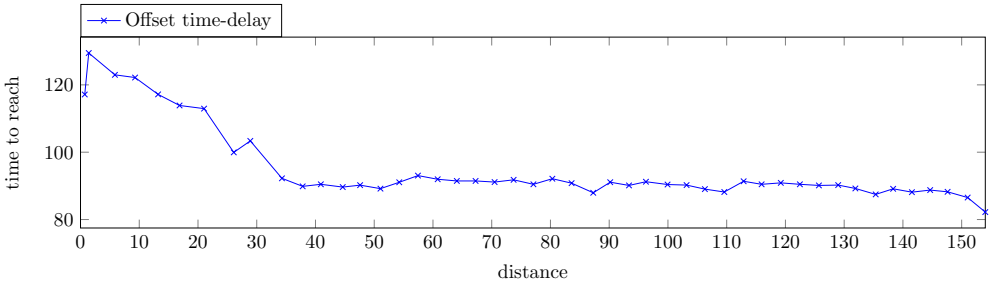
1 Tablet 200mm/s Epson

1.1 Information

...

1.2 Empirical Result





1.3 Data

Average display update time: 16.5909±1.02952ms
Average display update step: 3.40041±0.51156mm
Robot actual speed: 200mm/s
Calculated average robot speed: 194.428±0.283729mm/s
Calculated average object speed: 213.022±3.67788mm/s
The robot starts moving at time 655.736 and it takes 119.264 ms for the screen to respond
Estimated touch of display: 358 ms. Estimated time for screen response: 52ms (quite uncertain method estimate ±10 ms)
Median time delay: 90.4615
Median distance offset: 16.5575

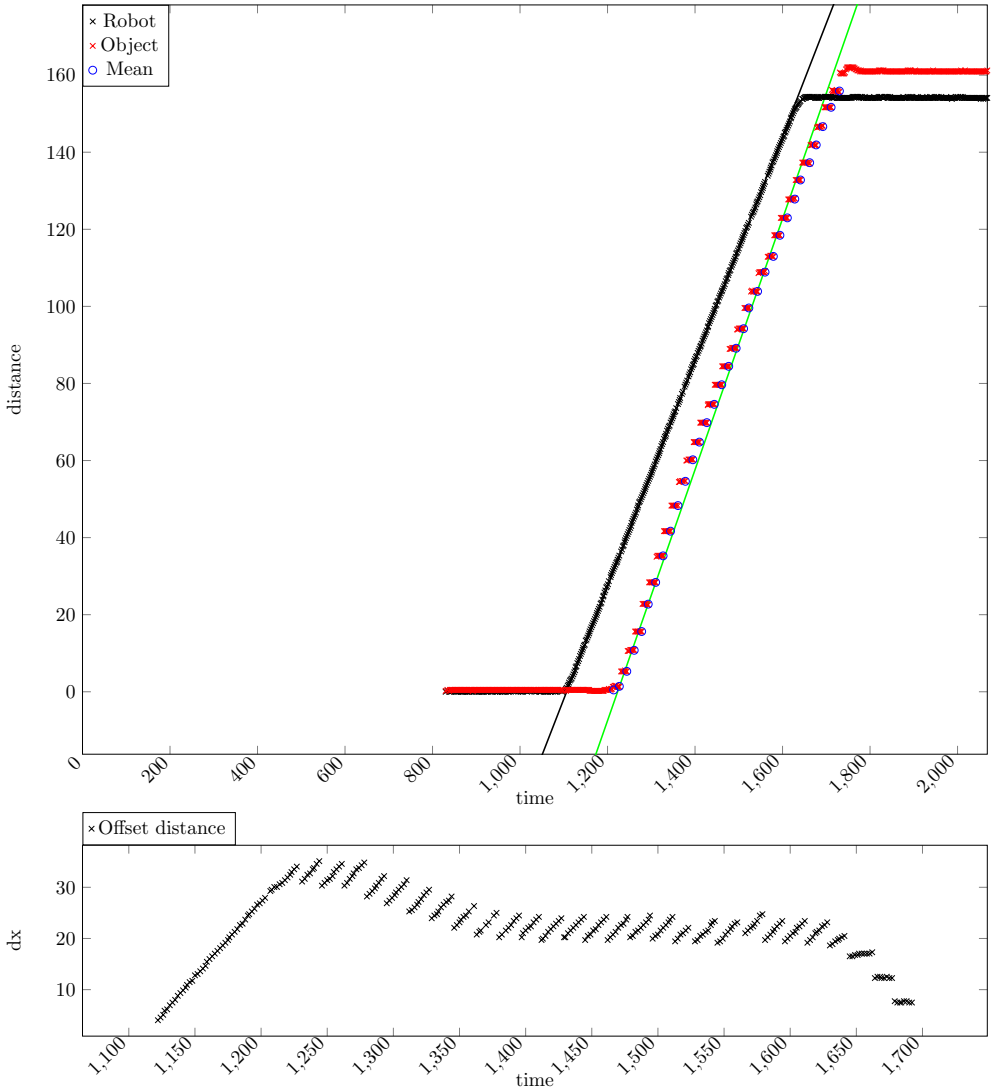
1.4 Warnings

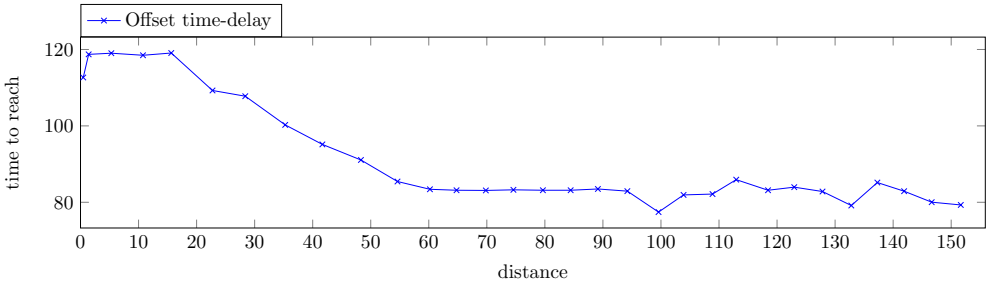
1 Tablet 300mm/s Epson

1.1 Information

...

1.2 Empirical Result





1.3 Data

Average display update time: 16.6897±1.78313ms
Average display update step: 5.18112±0.795778mm
Robot actual speed: 300mm/s
Calculated average robot speed: 292.237±0.403021mm/s
Calculated average object speed: 325.581±4.18037mm/s
The robot starts moving at time 1106.55 and it takes 107.454 ms for the screen to respond
Estimated touch of display: 774 ms. Estimated time for screen response: 56ms (quite uncertain method estimate ±10 ms)
Median time delay: 83.4017
Median distance offset: 22.3562

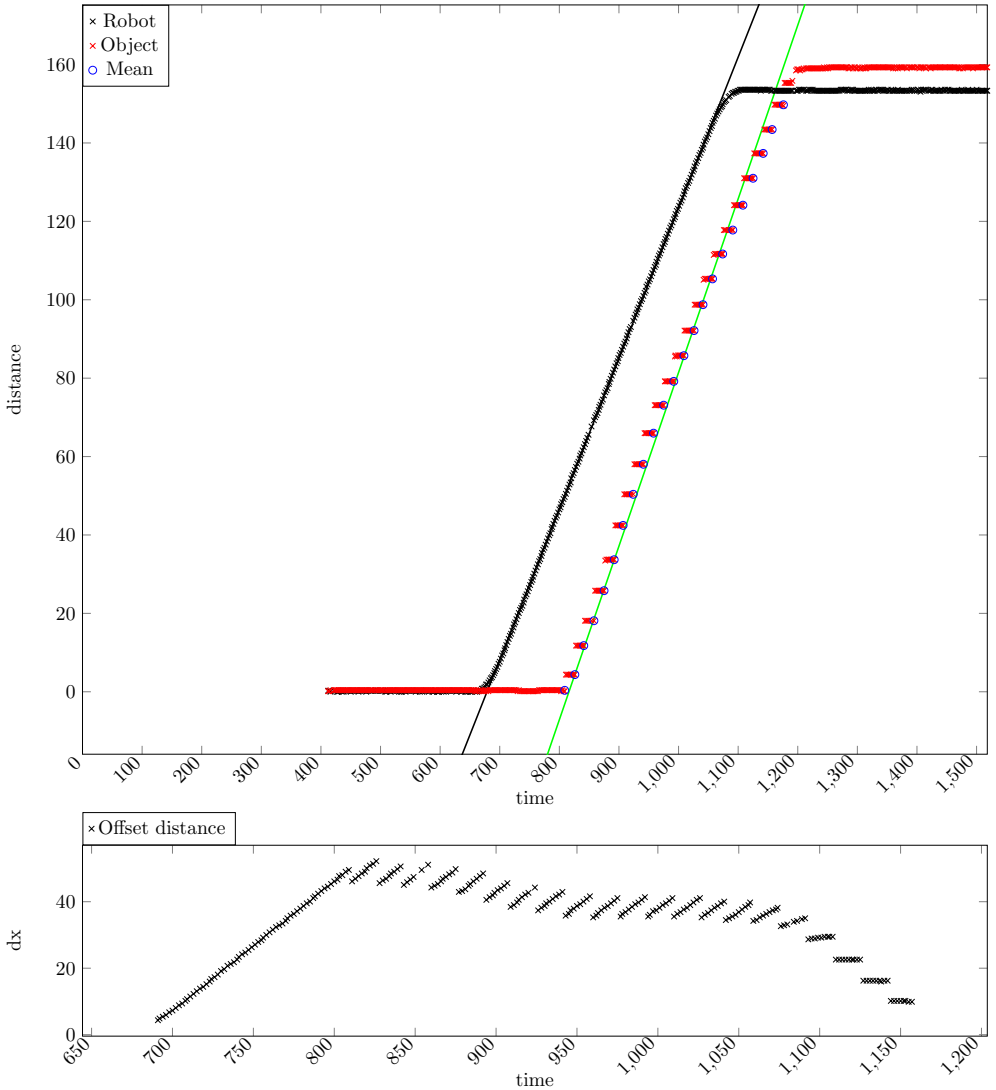
1.4 Warnings

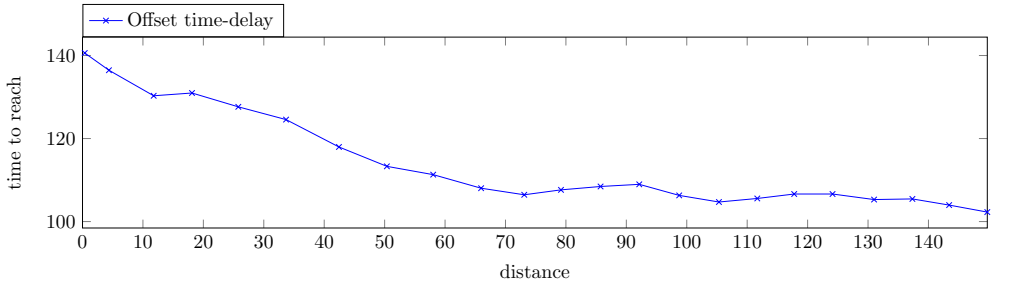
1 Tablet 400mm/s Epson

1.1 Information

...

1.2 Empirical Result





1.3 Data

Average display update time: 16.55±0.804674ms
Average display update step: 6.95246±0.766453mm
Robot actual speed: 400mm/s
Calculated average robot speed: 384.198±0.377877mm/s
Calculated average object speed: 442.918±4.71837mm/s
The robot starts moving at time 679.303 and it takes 129.697 ms for the screen to respond
Estimated touch of display: 352 ms. Estimated time for screen response: 60ms (quite uncertain method estimate ±10 ms)
Median time delay: 108.033
Median distance offset: 37.3935

1.4 Warnings

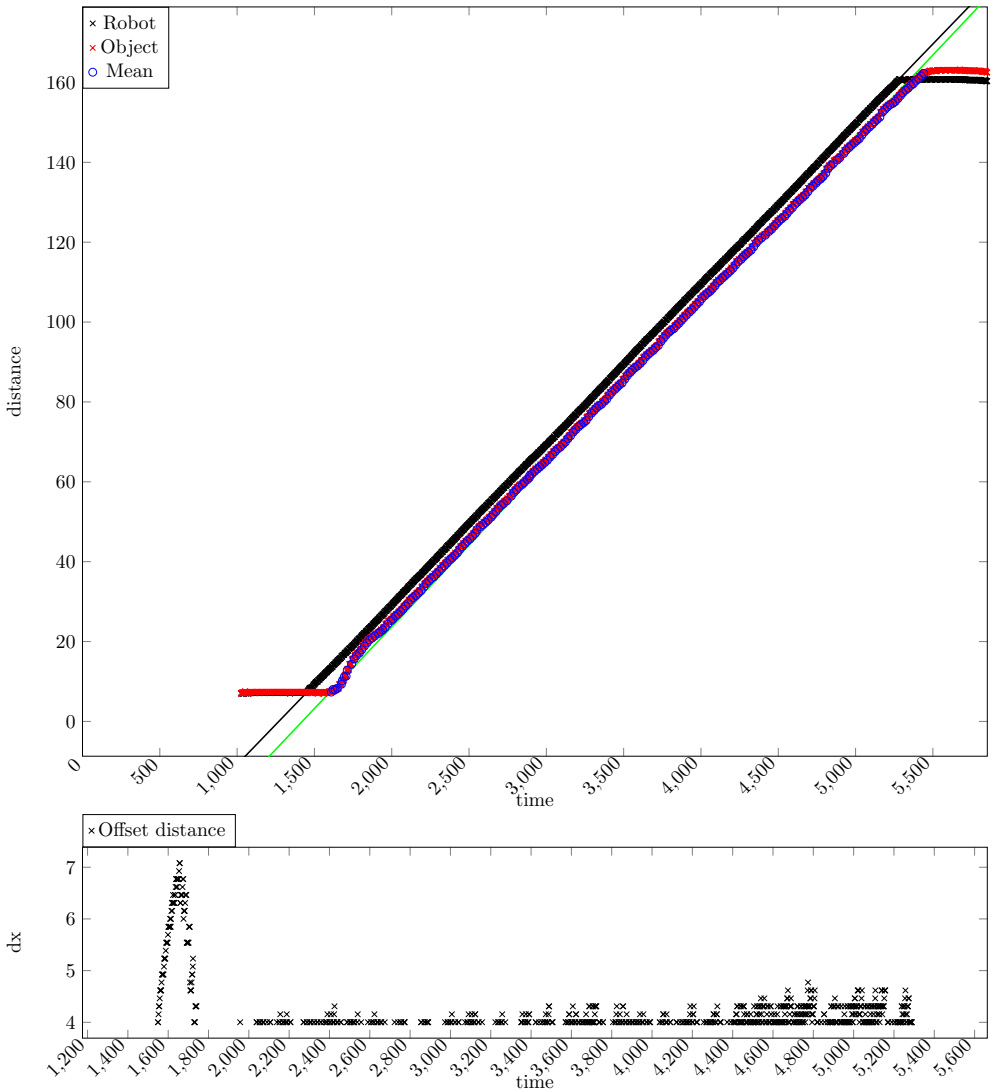
Swipe results Tablet 40-400 mm/s on K8200

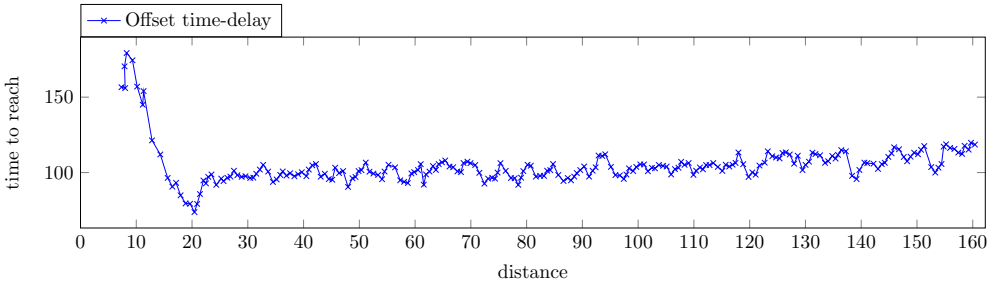
1 Tablet 40mm/s K8200

1.1 Information

...

1.2 Empirical Result





1.3 Data

Average display update time: 16.7788±2.86954ms
Average display update step: 0.680216±0.180981mm
Robot actual speed: 40mm/s
Calculated average robot speed: 40.0976±0.160314mm/s
Calculated average object speed: 40.9107±1.22853mm/s
The robot starts moving at time 1446.19 and it takes 162.814 ms for the screen to respond
Median time delay: 102.8
Median distance offset: 4.00121

1.4 Warnings

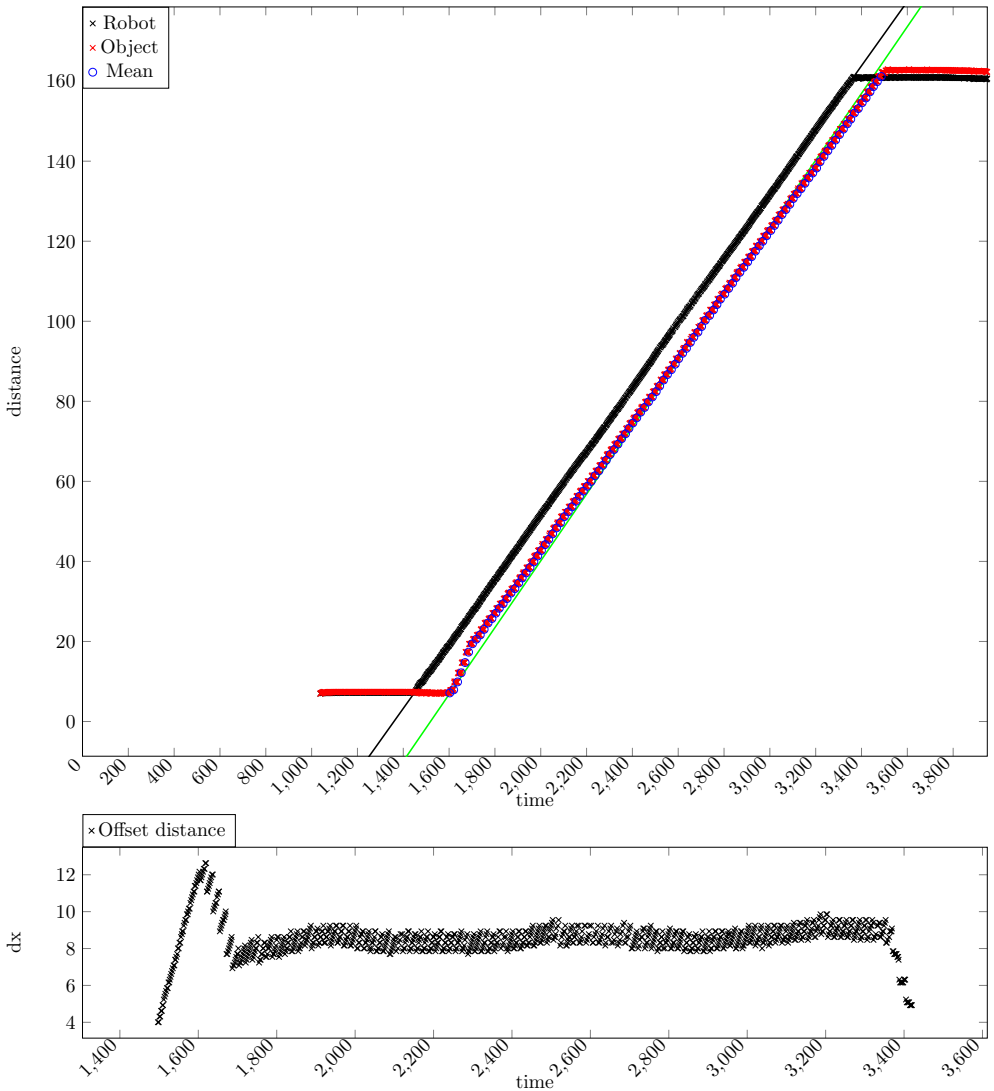
- WARNING 1: Init time is very large compared to average delay. Diff = 60.200001%
- WARNING 8: Suscpected framedrop(s) detected.

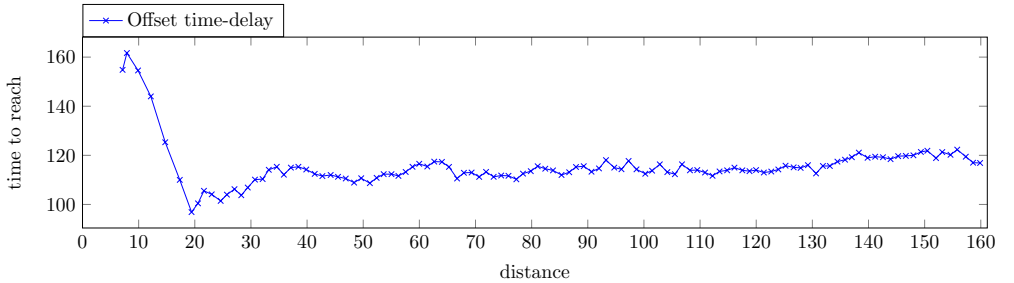
1 Tablet 80mm/s K8200

1.1 Information

...

1.2 Empirical Result





1.3 Data

Average display update time: 16.6545±0.835968ms
Average display update step: 1.36352±0.22788mm
Robot actual speed: 80mm/s
Calculated average robot speed: 80.2842±0.172944mm/s
Calculated average object speed: 83.4192±2.175mm/s
The robot starts moving at time 1445.61 and it takes 157.385 ms for the screen to respond
Median time delay: 114
Median distance offset: 8.62445

1.4 Warnings

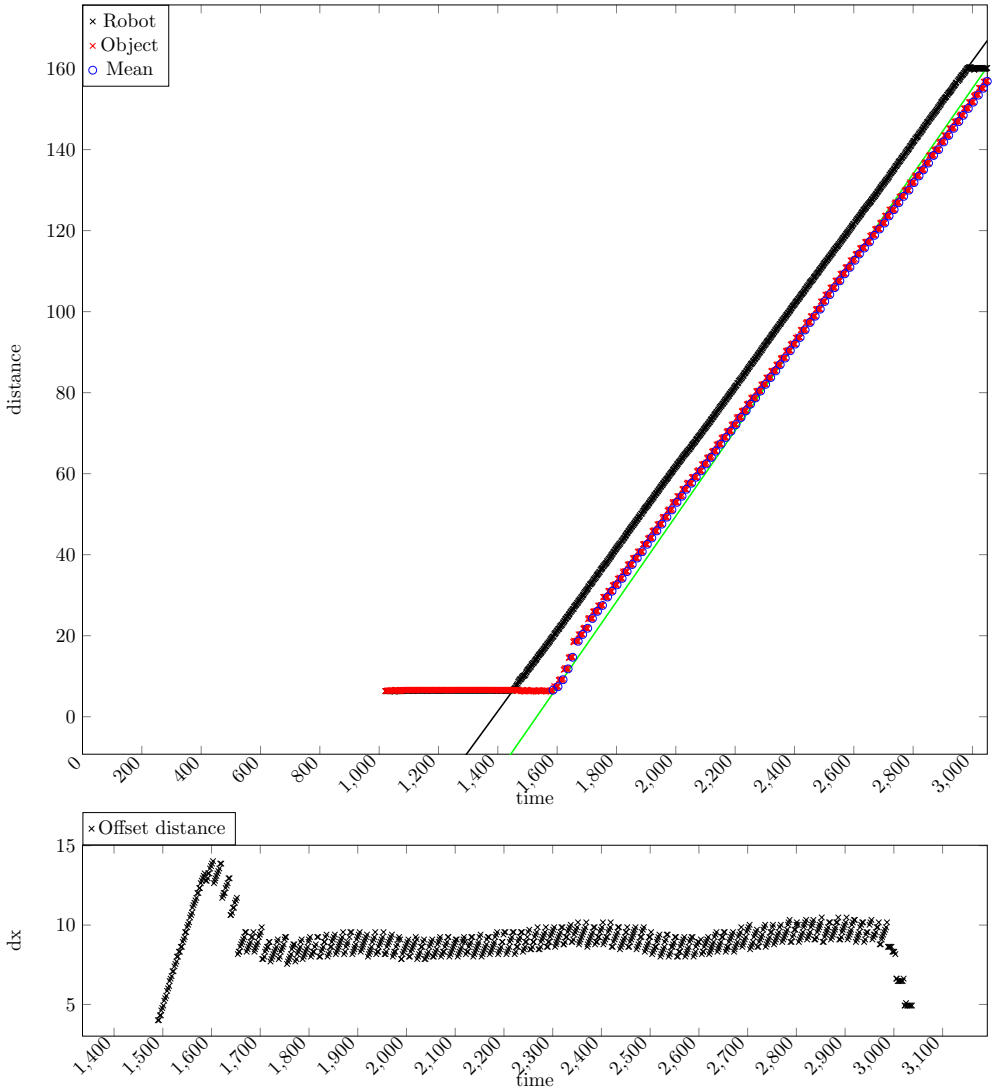
- WARNING 1: Init time is very large compared to average delay. Diff = 40.500000%

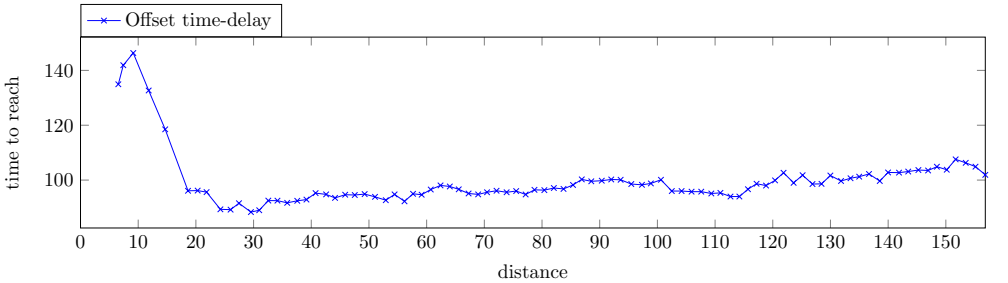
1 Tablet 100mm/s K8200

1.1 Information

...

1.2 Empirical Result





1.3 Data

Average display update time: 16.6706±0.787972ms
Average display update step: 1.71761±0.329538mm
Robot actual speed: 100mm/s
Calculated average robot speed: 100.327±0.17024mm/s
Calculated average object speed: 105.597±2.76391mm/s
The robot starts moving at time 1449.37 and it takes 136.625 ms for the screen to respond
Median time delay: 96.6665
Median distance offset: 8.93961

1.4 Warnings

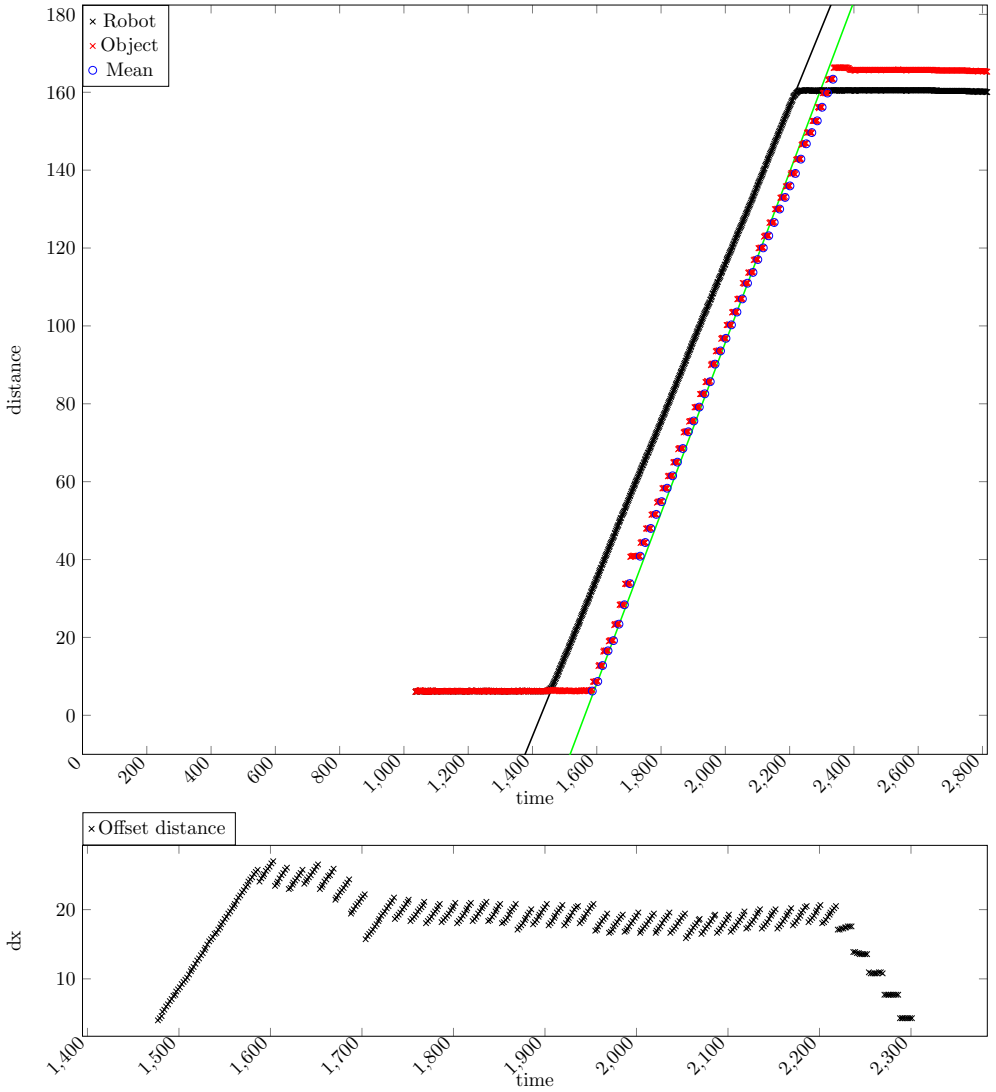
- WARNING 1: Init time is very large compared to average delay. Diff = 41.099998%
- WARNING 8: Suscpected framedrop(s) detected.

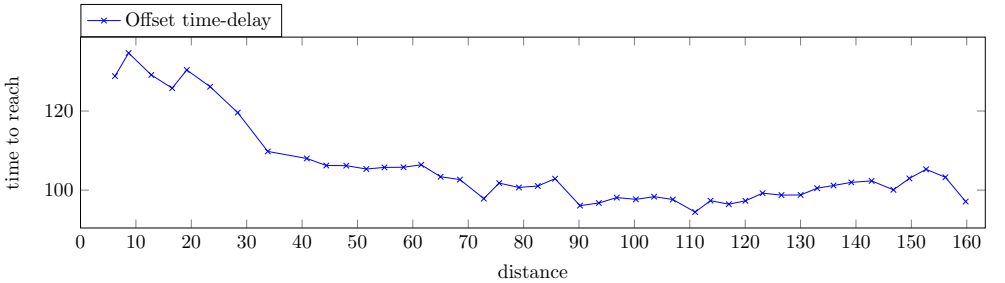
1 Tablet 200mm/s K8200

1.1 Information

...

1.2 Empirical Result





1.3 Data

Average display update time: 17.0238±2.44451ms
Average display update step: 3.59806±0.774199mm
Robot actual speed: 200mm/s
Calculated average robot speed: 202.234±0.184569mm/s
Calculated average object speed: 218.987±3.33281mm/s
The robot starts moving at time 1456.13 and it takes 129.872 ms for the screen to respond
Median time delay: 102.333
Median distance offset: 18.8106

1.4 Warnings

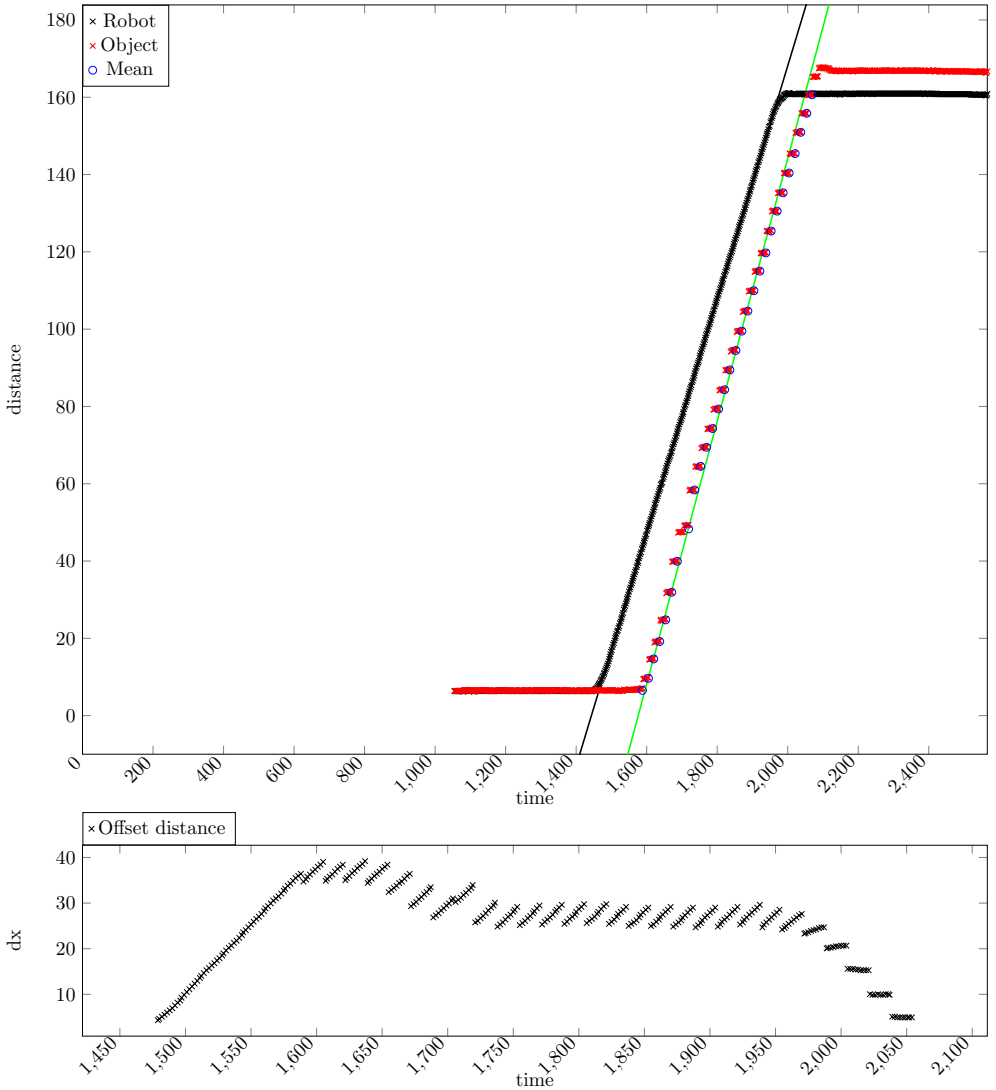
- WARNING 1: Init time is very large compared to average delay. Diff = 35.099998%
- WARNING 8: Suscpected framedrop(s) detected.

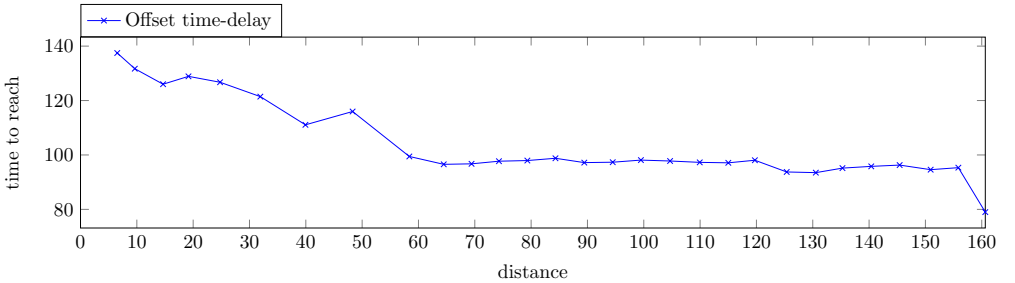
1 Tablet 300mm/s K8200

1.1 Information

...

1.2 Empirical Result





1.3 Data

Average display update time: 17.2692±3.0199ms
Average display update step: 5.62362±1.29385mm
Robot actual speed: 300mm/s
Calculated average robot speed: 301.934±0.239349mm/s
Calculated average object speed: 339.953±3.90835mm/s
The robot starts moving at time 1464.16 and it takes 123.843 ms for the screen to respond
Median time delay: 97.7037
Median distance offset: 27.0037

1.4 Warnings

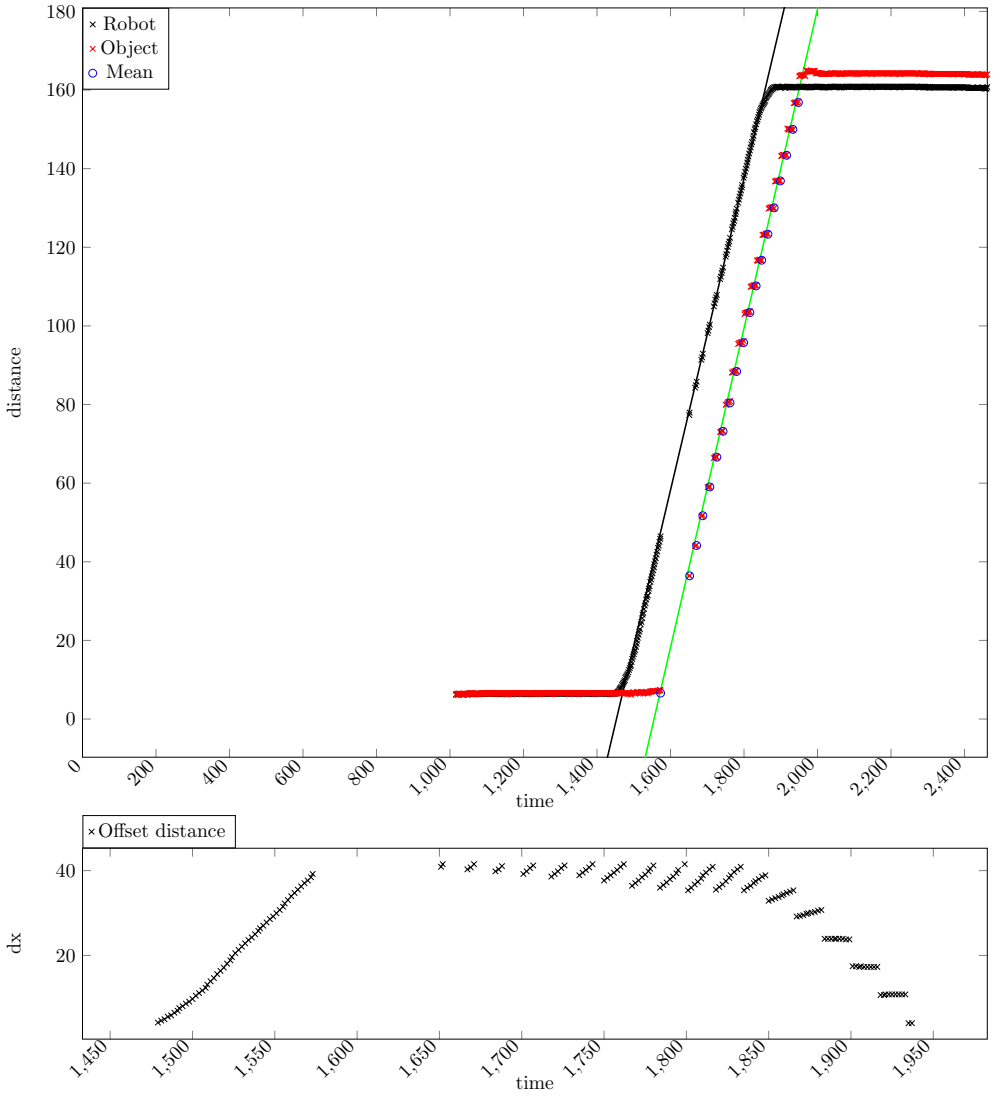
- WARNING 4: Time-steps found differs more than 10% from theoretical amount of time-steps. Diff = -13.000000%
- WARNING 8: Suscpected framedrop(s) detected.

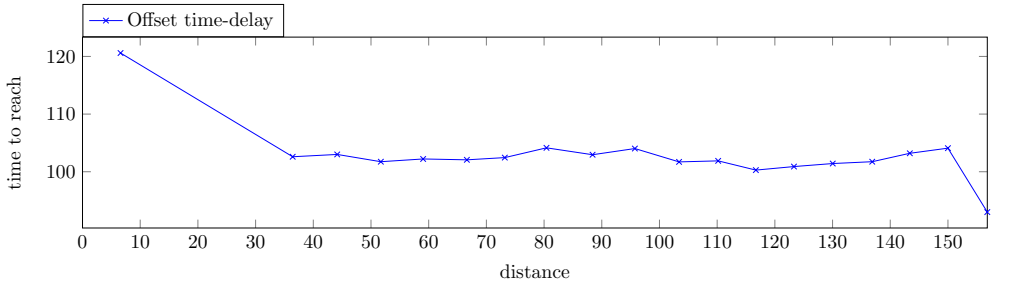
1 Tablet 400mm/s K8200

1.1 Information

...

1.2 Empirical Result





1.3 Data

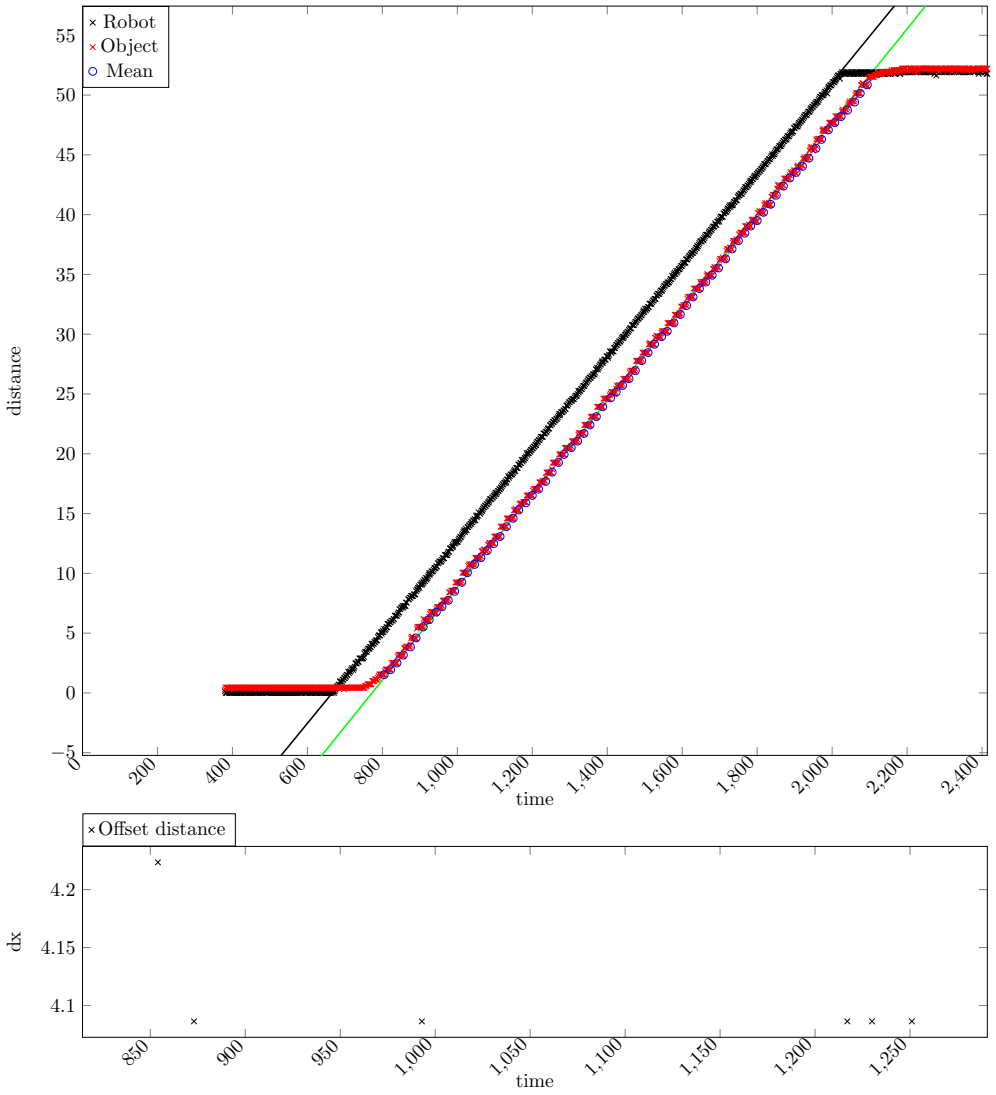
Average display update time: 17.5625±1.11629ms
Average display update step: 7.09799±0.487016mm
Robot actual speed: 400mm/s
Calculated average robot speed: 396.055±1.22859mm/s
Calculated average object speed: 405.945±1.83158mm/s
The robot starts moving at time 1468.53 and it takes 104.471 ms for the screen to respond
Median time delay: 102.215
Median distance offset: 33.8028

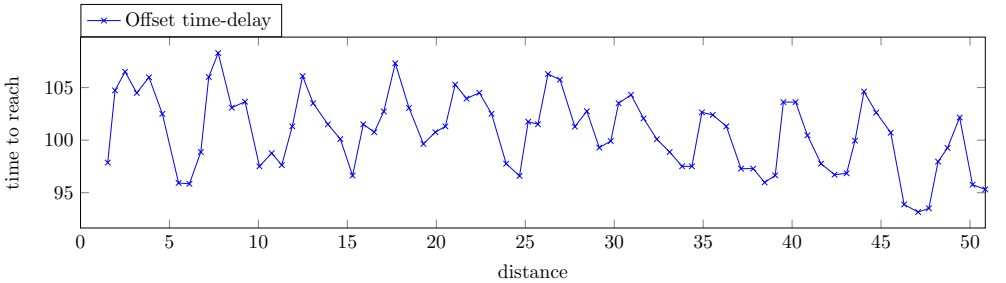
1.4 Warnings

- WARNING 4: Time-steps found differs more than 10% from theoretical amount of time-steps. Diff = -22.799999%
- WARNING 8: Suscpected framedrop(s) detected.

Swipe results Small Phone 40-400 mm/s on Epson

1 Small Phone 40mm/s Epson



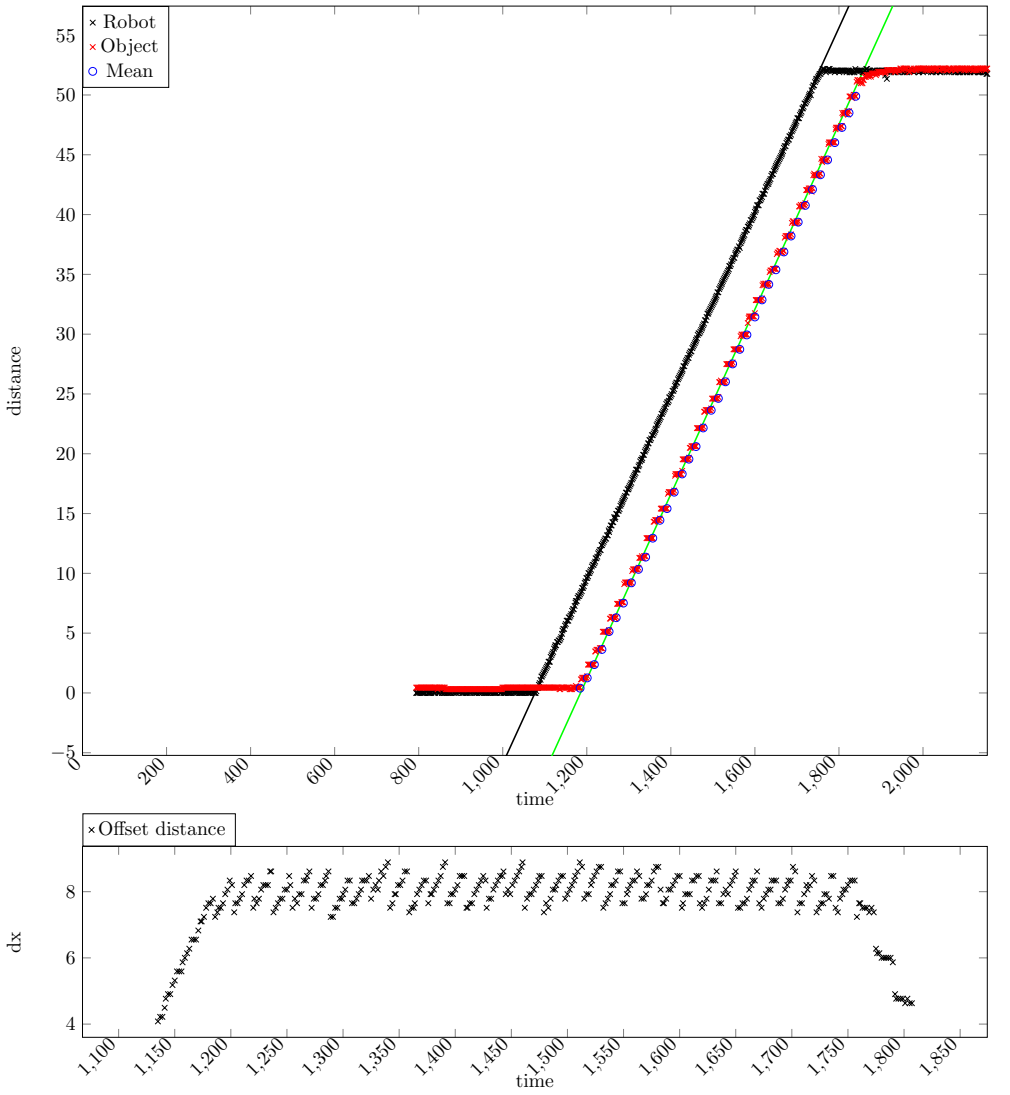


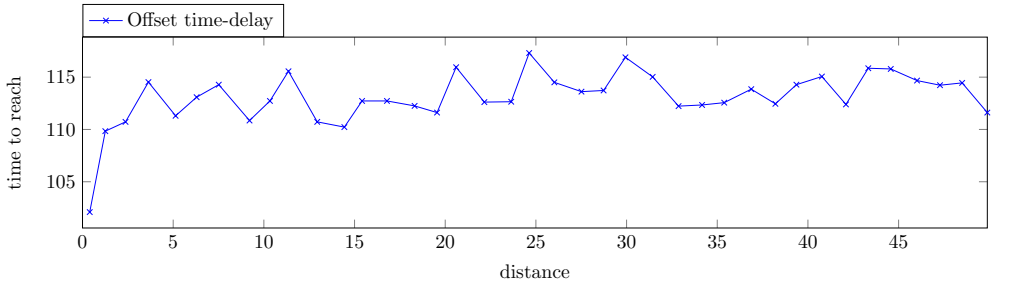
1.1 Data

Average display update time: 17.1667±1.2693ms
Average display update step: 0.661616±0.10438mm
Robot actual speed: 40mm/s
Calculated average robot speed: 38.3053±0.0777466mm/s
Calculated average object speed: 38.9063±0.271198mm/s
The robot starts moving at time 669.935 and it takes 135.065 ms for the screen to respond
Estimated touch of display: 293 ms. Estimated time for screen response: 87ms (quite uncertain method estimate ±10 ms)
Median time delay: 101.31
Median distance offset: 4.08626

1.2 Warnings

1 Small Phone 80mm/s Epson



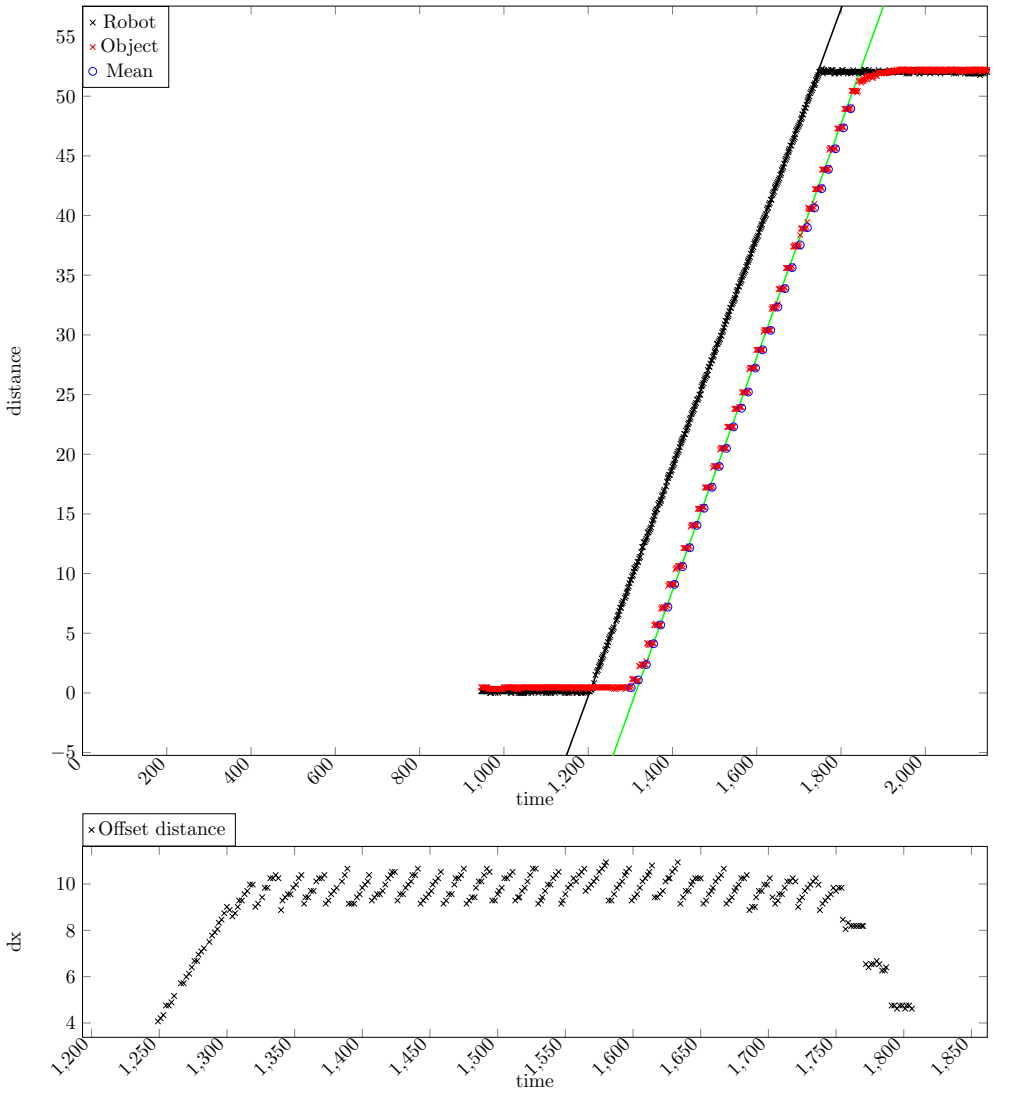


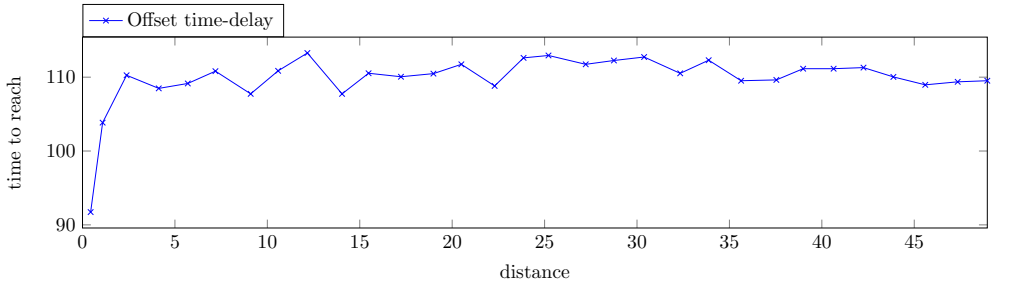
1.1 Data

Average display update time: 17.3143±0.820154ms
Average display update step: 1.31805±0.176524mm
Robot actual speed: 80mm/s
Calculated average robot speed: 76.9019±0.0883197mm/s
Calculated average object speed: 77.3149±0.400477mm/s
The robot starts moving at time 1076.73 and it takes 107.273 ms for the screen to respond
Estimated touch of display: 706 ms. Estimated time for screen response: 88ms (quite uncertain method estimate ±10 ms)
Median time delay: 112.725
Median distance offset: 7.92556

1.2 Warnings

1 Small Phone 100mm/s Epson



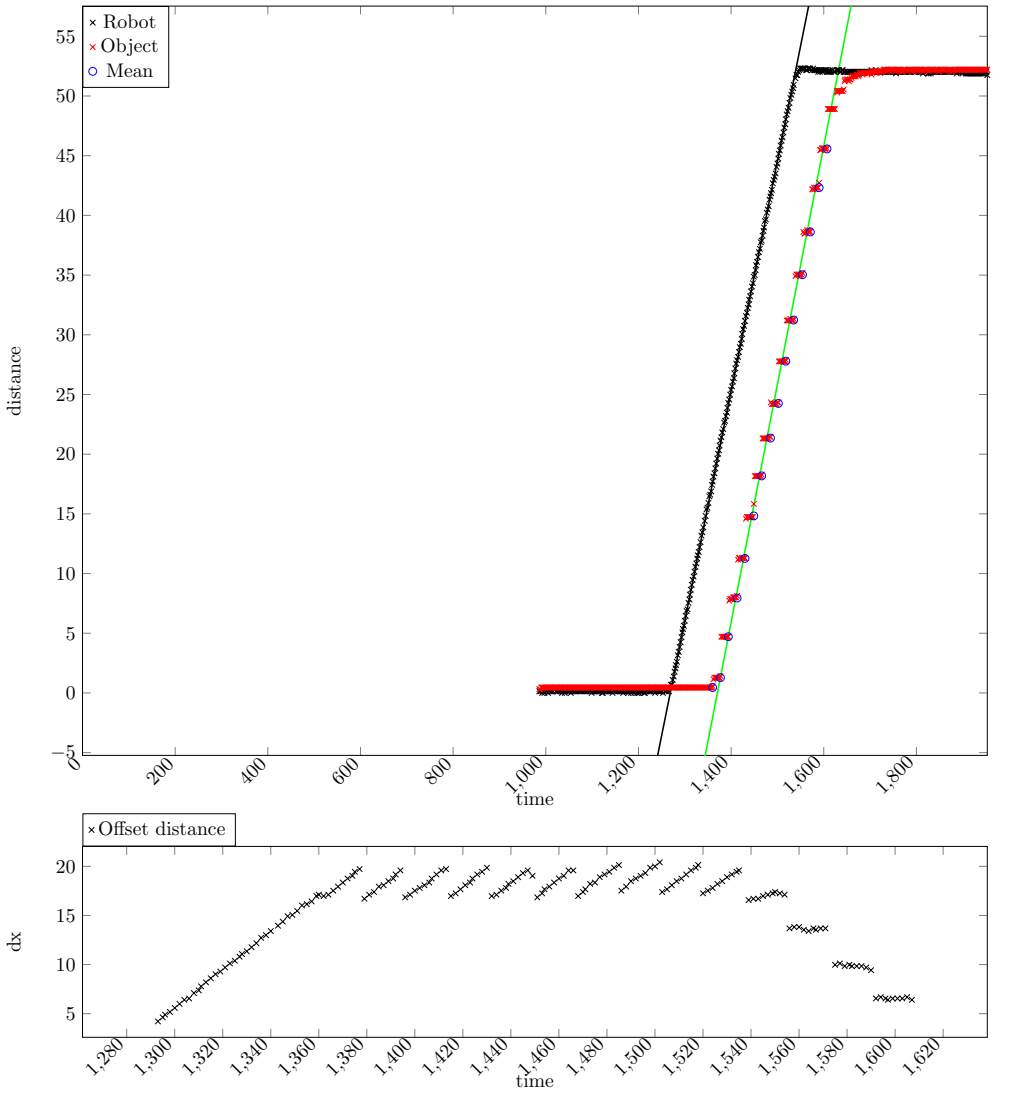


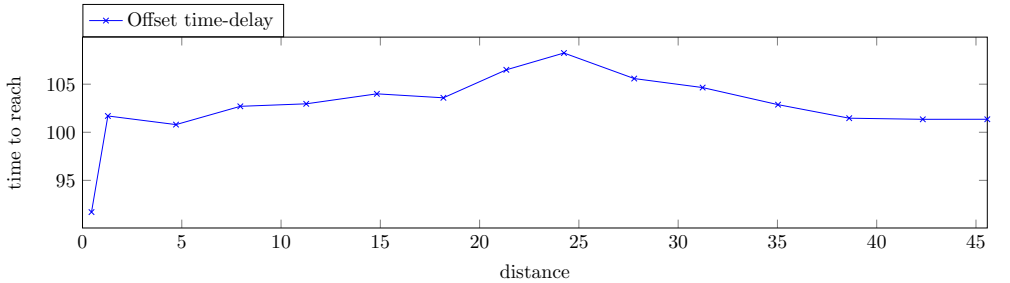
1.1 Data

Average display update time: 17.3333±0.942809ms
Average display update step: 1.66537±0.169588mm
Robot actual speed: 100mm/s
Calculated average robot speed: 96.0043±0.0921952mm/s
Calculated average object speed: 97.8303±0.547998mm/s
The robot starts moving at time 1204.37 and it takes 97.6322 ms for the screen to respond
Estimated touch of display: 828 ms. Estimated time for screen response: 118ms (quite uncertain method estimate ±10 ms)
Median time delay: 110.48
Median distance offset: 9.69879

1.2 Warnings

1 Small Phone 200mm/s Epson





1.1 Data

Average display update time: 17.75±1.08972ms
Average display update step: 3.42036±0.235495mm
Robot actual speed: 200mm/s
Calculated average robot speed: 192.619±0.110892mm/s
Calculated average object speed: 199.479±1.09512mm/s
The robot starts moving at time 1269.34 and it takes 90.6558 ms for the screen to respond
Estimated touch of display: 888 ms. Estimated time for screen response: 98ms (quite uncertain method estimate ±10 ms)
Median time delay: 102.871
Median distance offset: 17.2544

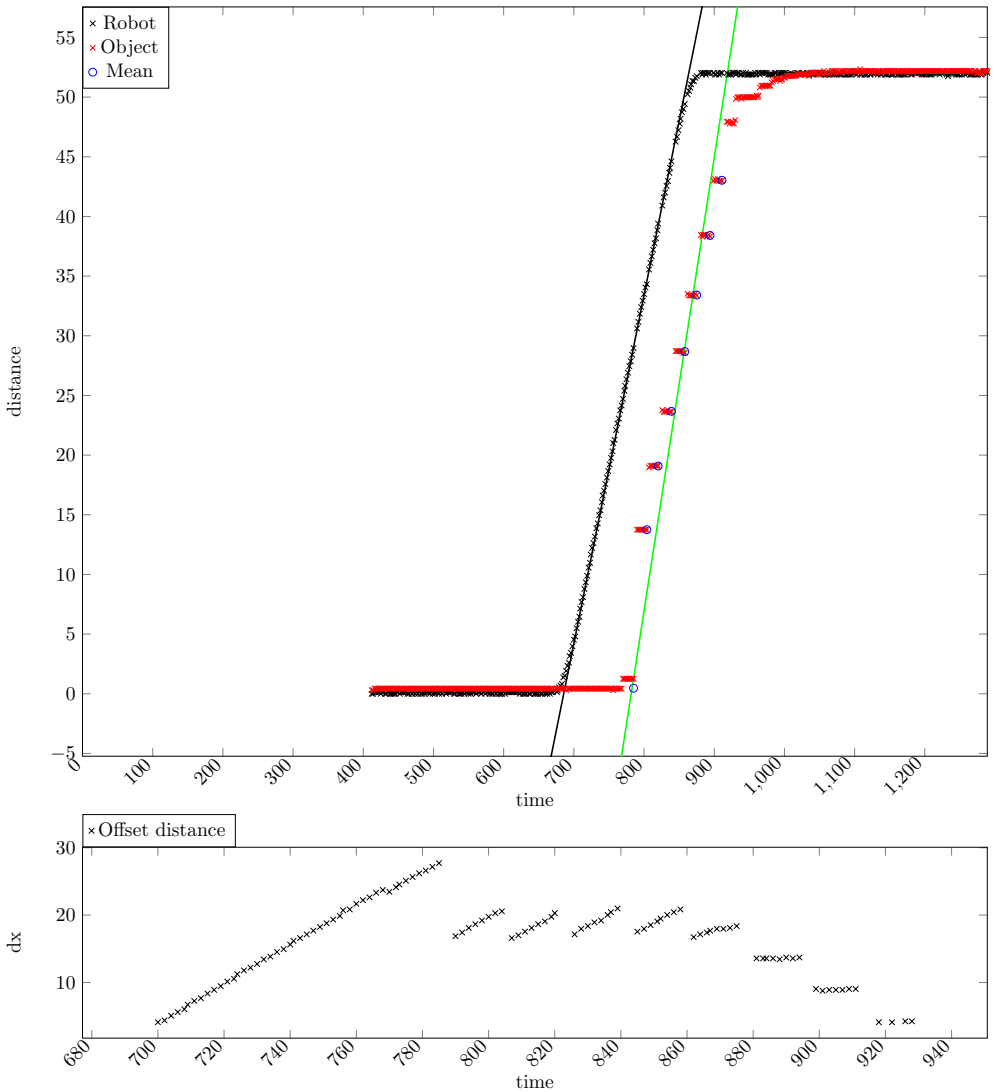
1.2 Warnings

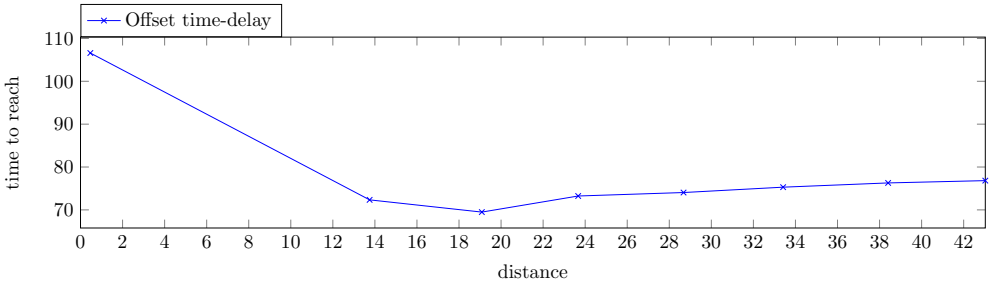
1 Small Phone 300mm/s Epson

1.1 Information

...

1.2 Empirical Result





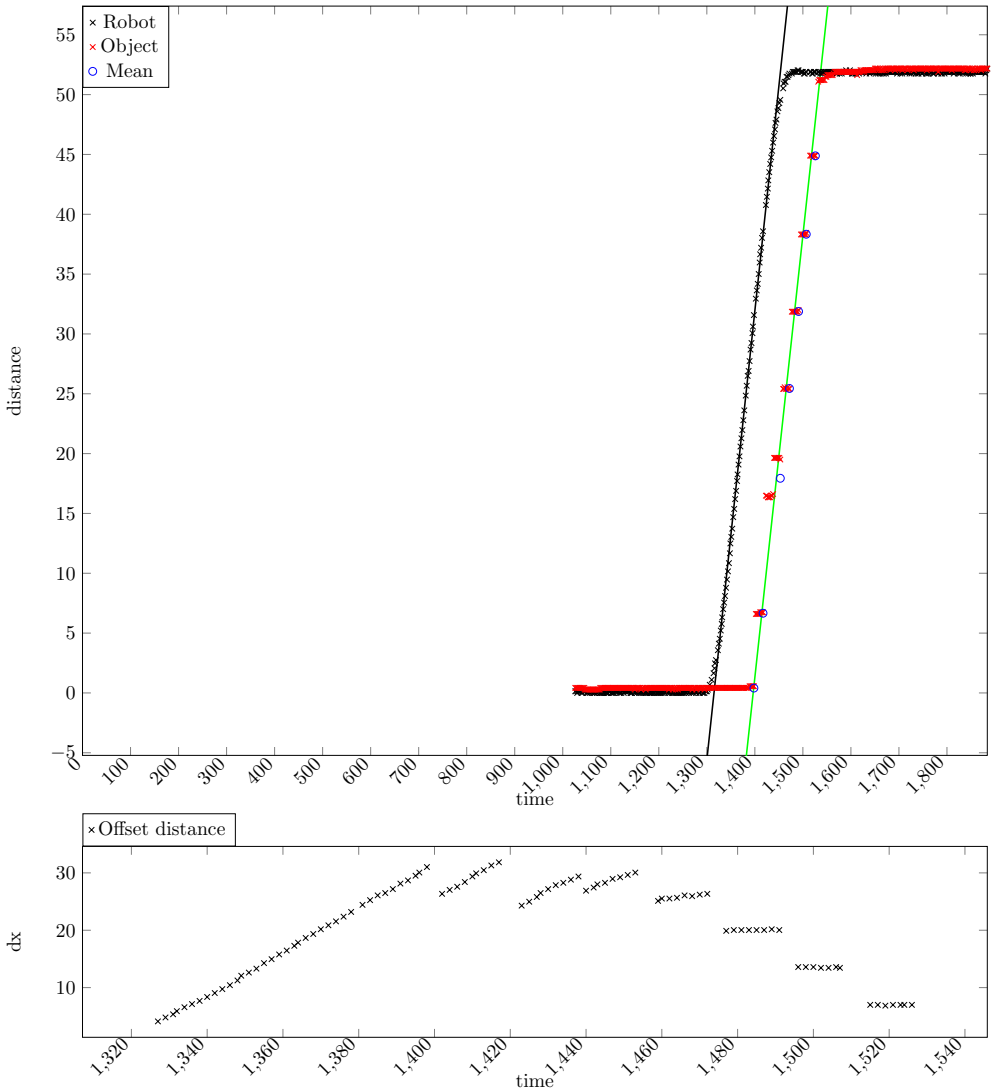
1.3 Data

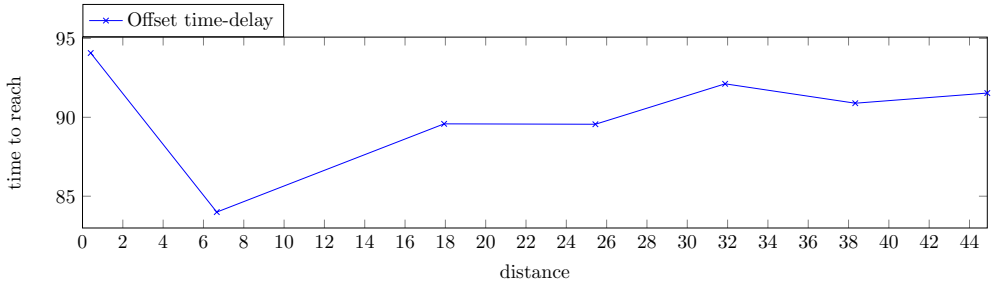
Average display update time: 18±1.26491ms
Average display update step: 4.93178±0.258792mm
Robot actual speed: 300mm/s
Calculated average robot speed: 291.908±0.212273mm/s
Calculated average object speed: 379.78±4.87006mm/s
The robot starts moving at time 685.54 and it takes 99.46 ms for the screen to respond
Estimated touch of display: 305 ms. Estimated time for screen response: 107ms (quite uncertain method estimate ±10 ms)
Median time delay: 75.299
Median distance offset: 17.551

1.4 Warnings

- WARNING 1: Init time is very large compared to average delay. Diff = 34.299999%
- WARNING 4: Time-steps found differs more than 10% from theoretical amount of time-steps. Diff = -22.600000%
- WARNING 8: Suscpected framedrop(s) detected.

1 Small Phone 400mm/s Epson





1.1 Data

Average display update time: 22.5±7.88987ms
Average display update step: 7.91791±1.98711mm
Robot actual speed: 400mm/s
Calculated average robot speed: 375.367±0.348306mm/s
Calculated average object speed: 369.411±2.41071mm/s
The robot starts moving at time 1315.3 and it takes 82.7036 ms for the screen to respond
Estimated touch of display: 920 ms. Estimated time for screen response: 106ms (quite uncertain method estimate ±10 ms)
Median time delay: 90.8925
Median distance offset: 21.544

1.2 Warnings

- WARNING 2: Calculated velocities differs more than 5% from actual robot velocity. Diff = 6.500000%
- WARNING 4: Time-steps found differs more than 10% from theoretical amount of time-steps. Diff = -10.300000%
- WARNING 6: Average display update is high. (60 Hz = 16.666 ms)

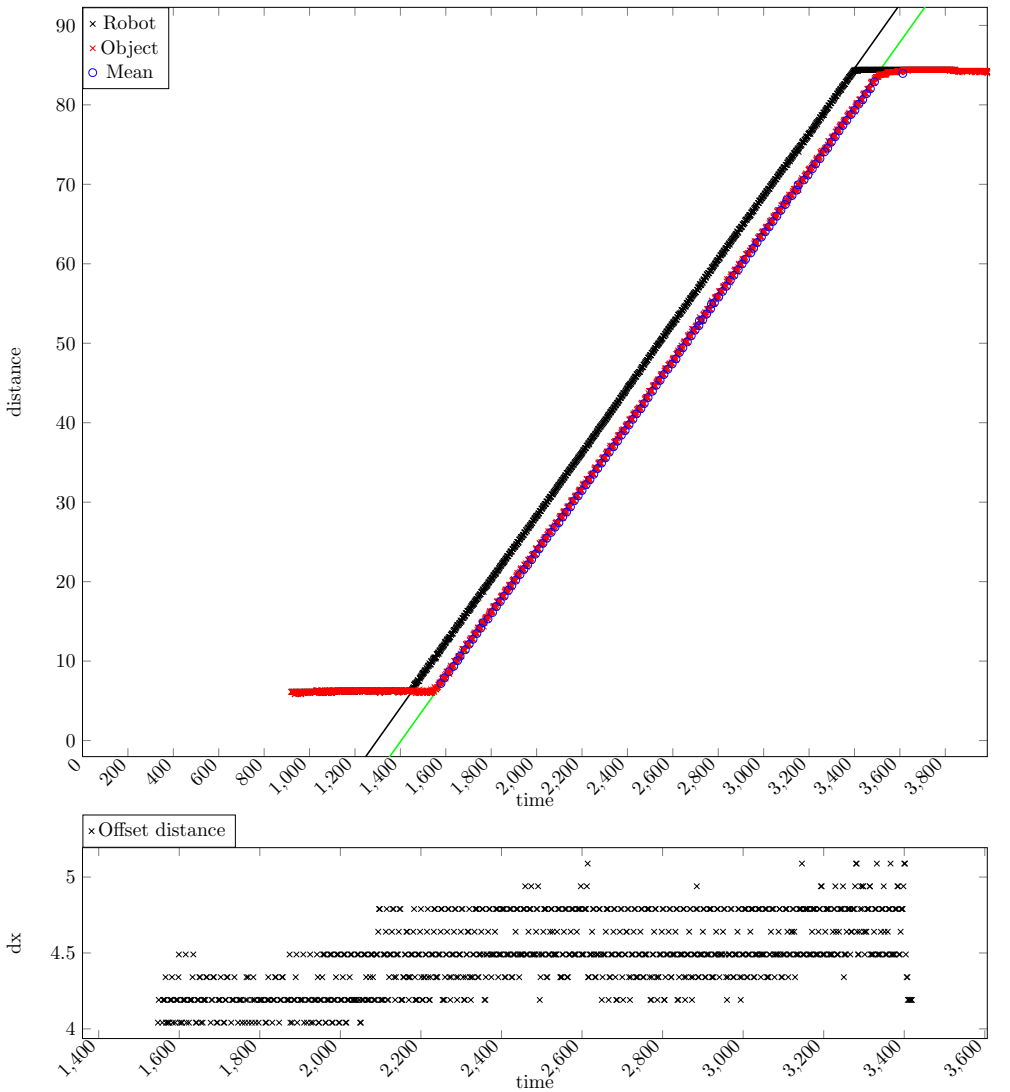
Swipe results Small Phone 40-400 mm/s on K8200

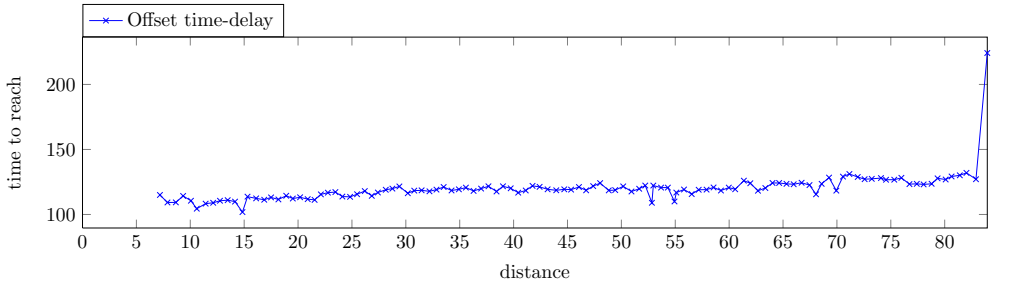
1 Z1 COMPACT-2014-9-3-16:28:52-40mms.mkv

1.1 Information

...

1.2 Empirical Result





1.3 Data

Average display update time: 16.8829±3.08364ms
Average display update step: 0.668853±0.100558mm
Robot actual speed: 40mm/s
Calculated average robot speed: 40.2357±0.134103mm/s
Calculated average object speed: 40.0098±0.275348mm/s
The robot starts moving at time 1449.97 and it takes 129.035 ms for the screen to respond
Median time delay: 119.111
Median distance offset: 4.4901

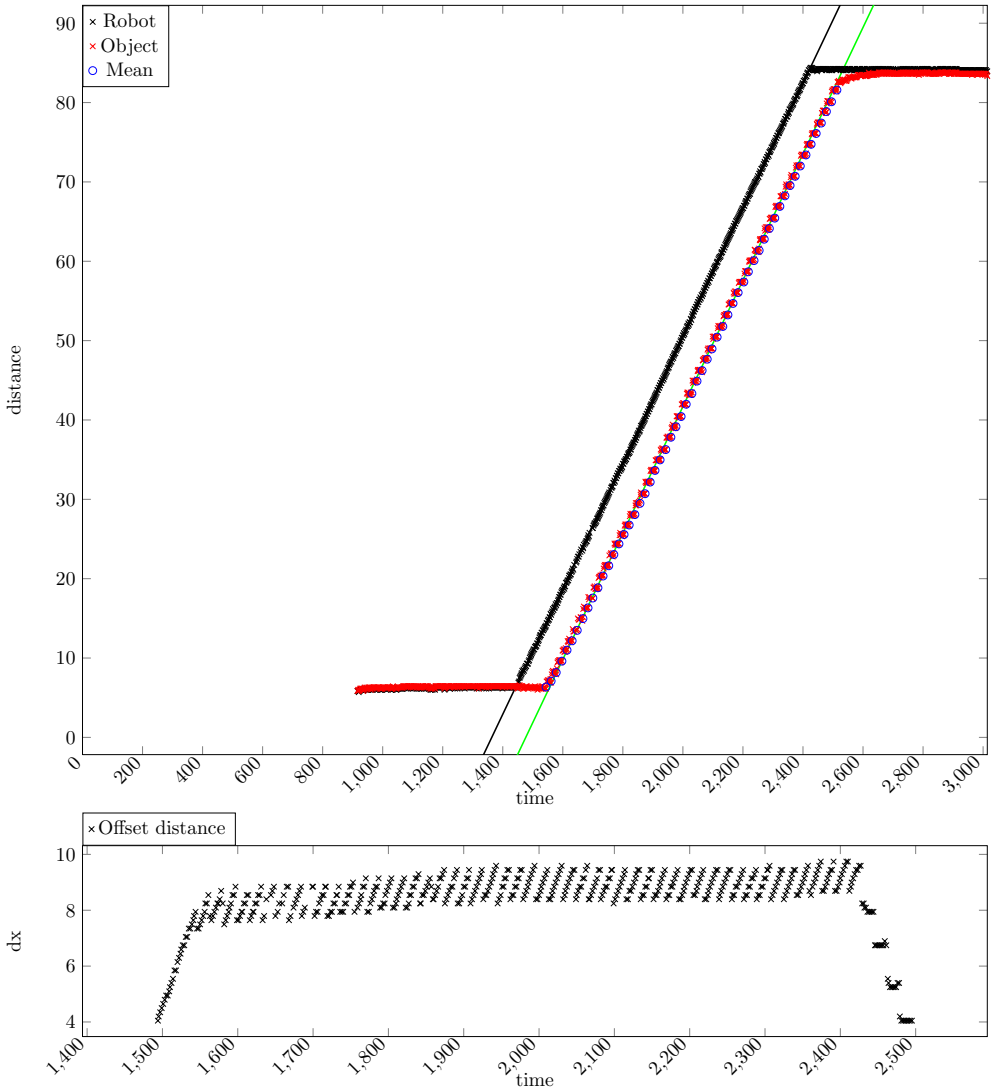
1.4 Warnings

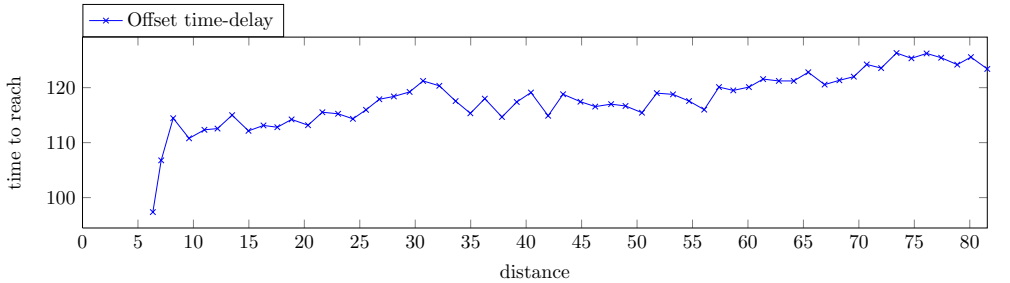
1 Z1 COMPACT-2014-9-3-16:29:33-80mms.mkv

1.1 Information

...

1.2 Empirical Result





1.3 Data

Average display update time: 17.3019±0.90251ms
Average display update step: 1.35709±0.0947133mm
Robot actual speed: 80mm/s
Calculated average robot speed: 79.5288±0.257572mm/s
Calculated average object speed: 79.6372±0.456885mm/s
The robot starts moving at time 1436.81 and it takes 107.192 ms for the screen to respond
Median time delay: 117.889
Median distance offset: 8.69301

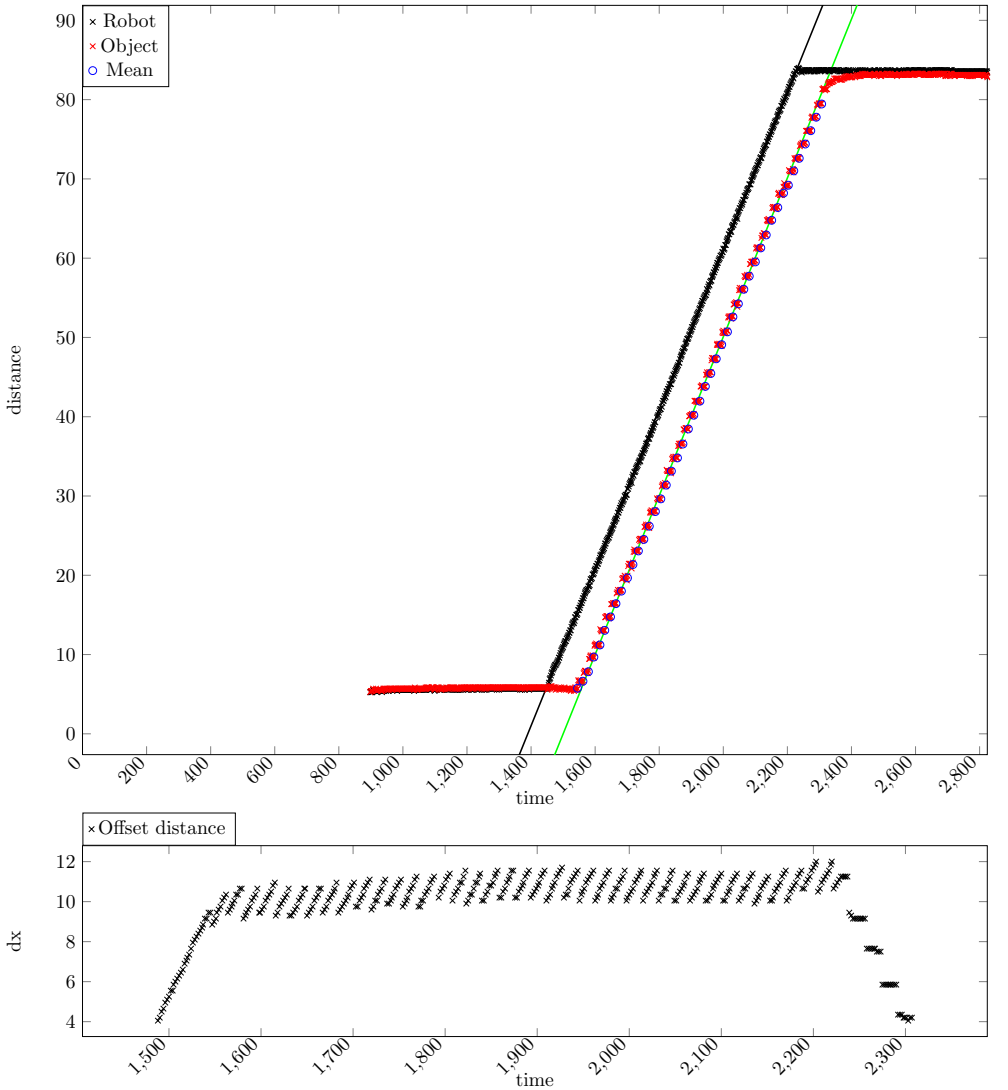
1.4 Warnings

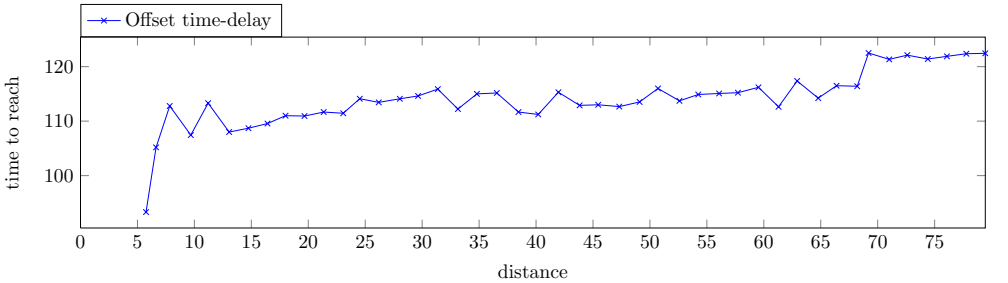
1 Z1 COMPACT-2014-9-3-16:29:51-100mms.mkv

1.1 Information

...

1.2 Empirical Result





1.3 Data

Average display update time: 17.3415±0.978046ms
Average display update step: 1.70601±0.152954mm
Robot actual speed: 100mm/s
Calculated average robot speed: 99.8365±0.262249mm/s
Calculated average object speed: 100.184±0.596665mm/s
The robot starts moving at time 1442.41 and it takes 102.59 ms for the screen to respond
Median time delay: 114.1
Median distance offset: 10.3519

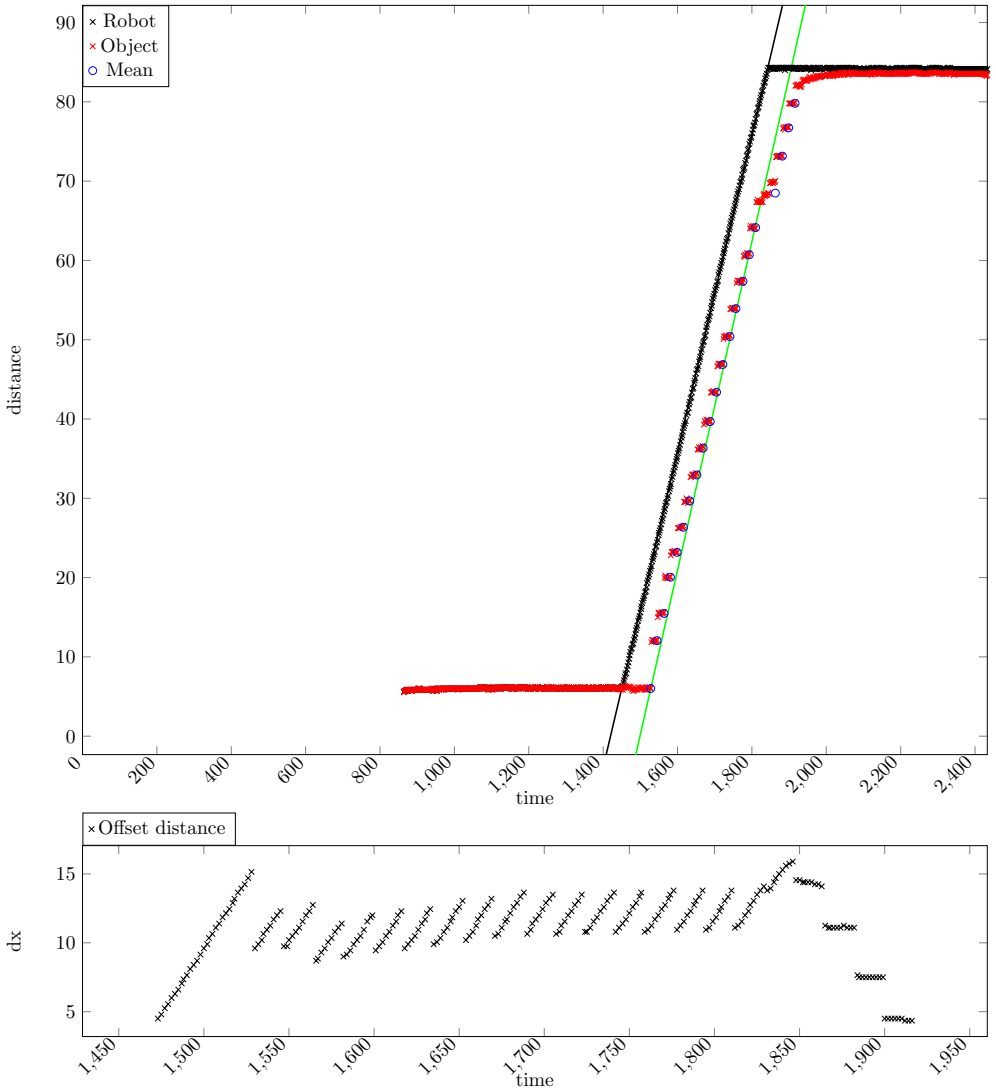
1.4 Warnings

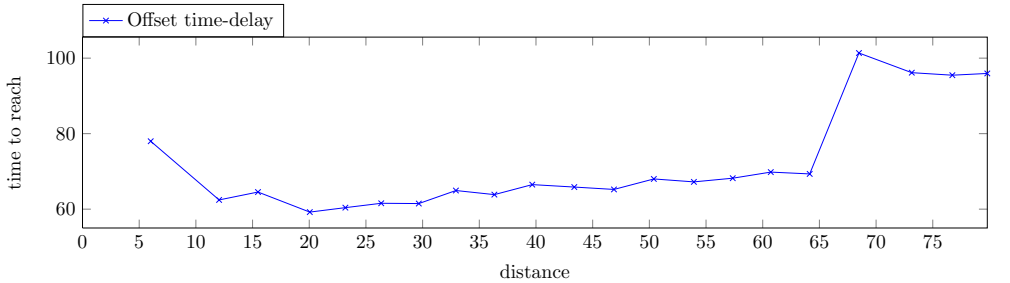
1 Z1 COMPACT-2014-9-3-16:30:19-200mms.mkv

1.1 Information

...

1.2 Empirical Result





1.3 Data

Average display update time: 19.6667±8.14453ms
Average display update step: 3.5932±0.440858mm
Robot actual speed: 200mm/s
Calculated average robot speed: 199.303±0.346694mm/s
Calculated average object speed: 207.256±3.2508mm/s
The robot starts moving at time 1447.67 and it takes 80.3293 ms for the screen to respond
Median time delay: 66.5
Median distance offset: 11.399

1.4 Warnings

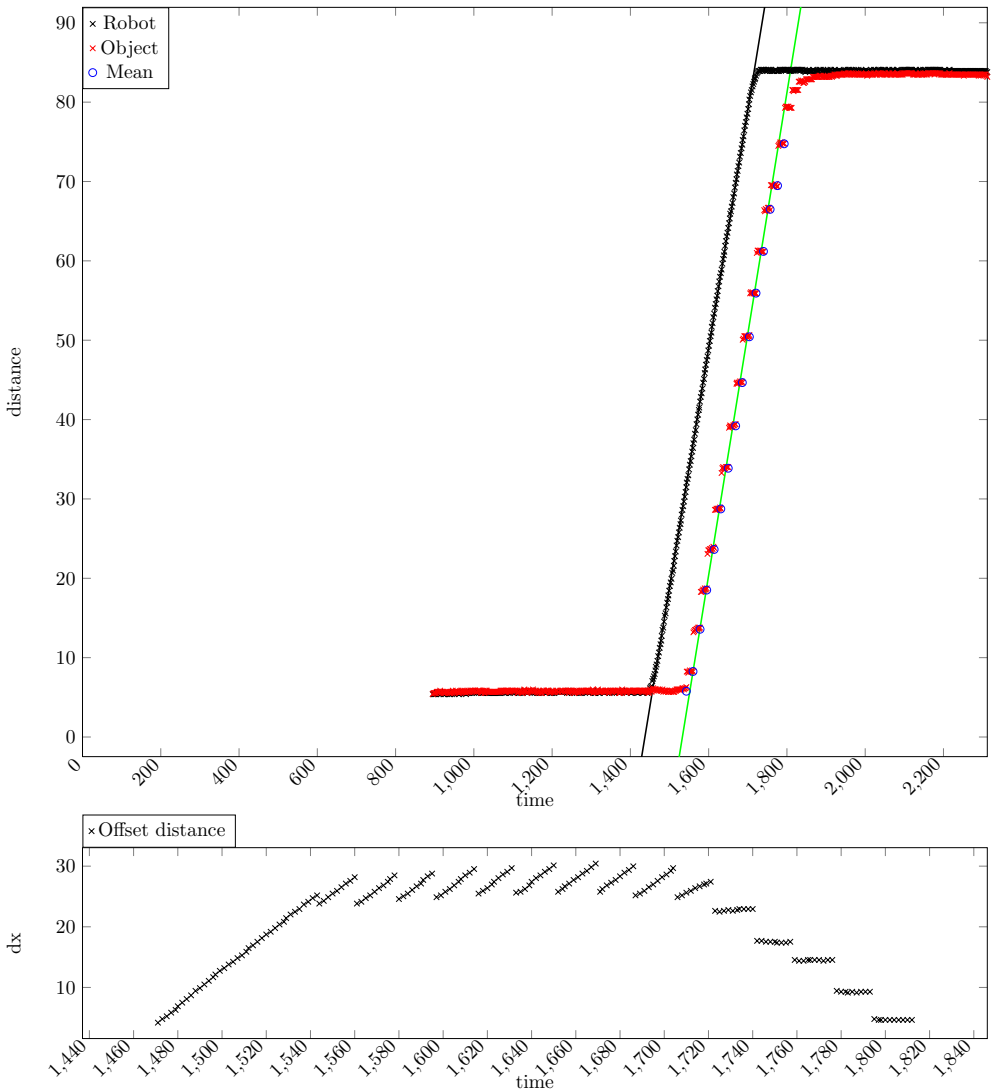
- WARNING 4: Time-steps found differs more than 10% from theoretical amount of time-steps. Diff = -16.100000%
- WARNING 6: Average display update is high. (60 Hz = 16.666 ms)

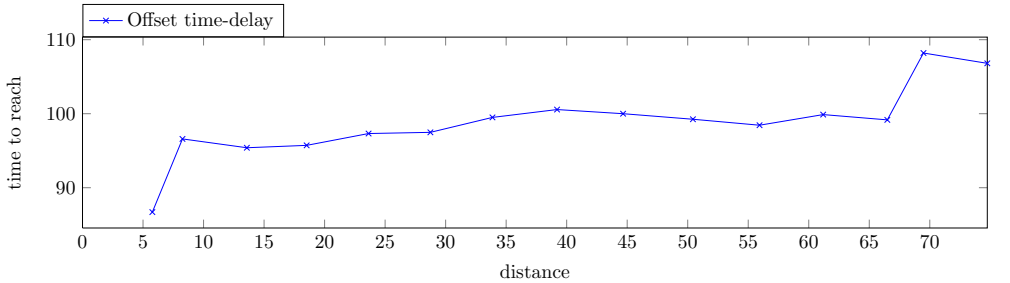
1 Z1 COMPACT-2014-9-3-16:30:44-300mms.mkv

1.1 Information

...

1.2 Empirical Result





1.3 Data

Average display update time: 18±0.912871ms
Average display update step: 5.10166±0.670579mm
Robot actual speed: 300mm/s
Calculated average robot speed: 300.599±0.250785mm/s
Calculated average object speed: 303.96±2.00015mm/s
The robot starts moving at time 1455.05 and it takes 87.9542 ms for the screen to respond
Median time delay: 99.1666
Median distance offset: 24.7284

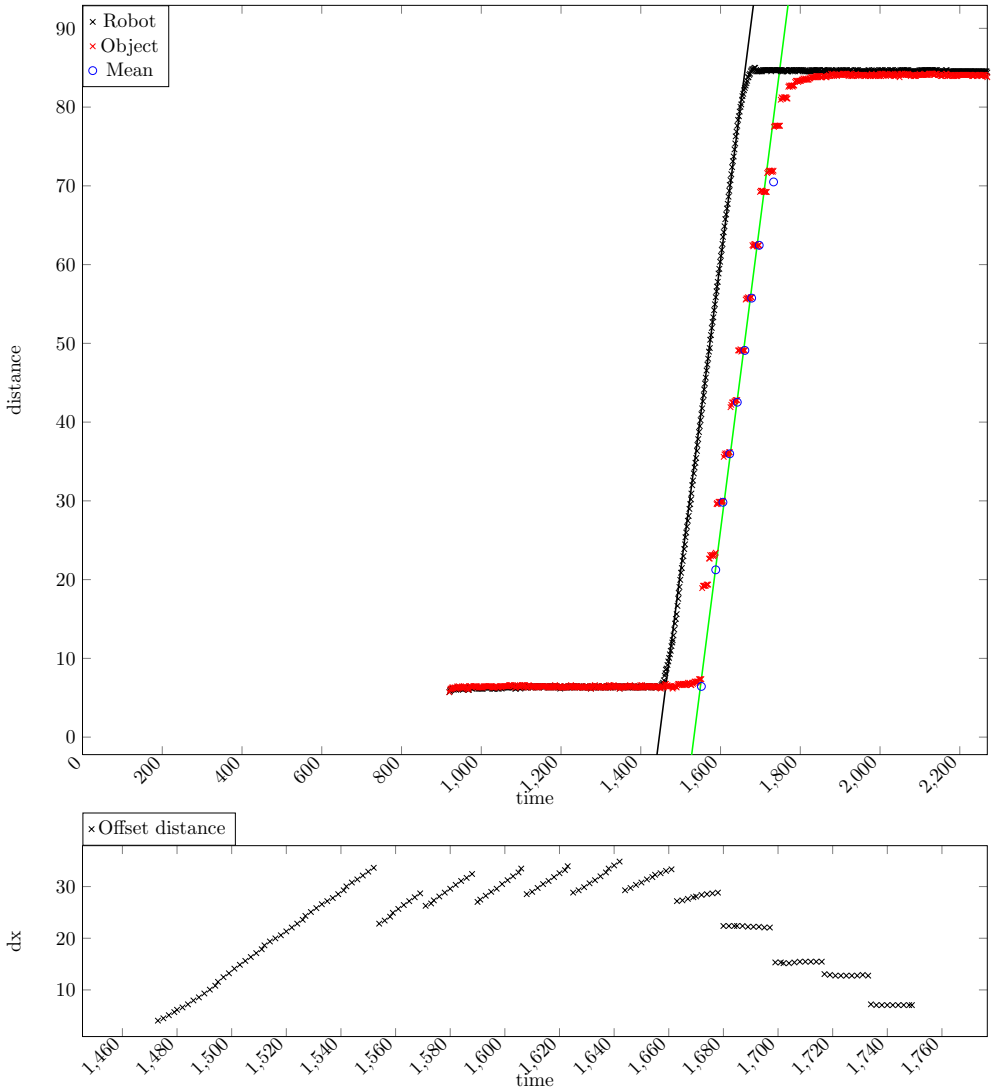
1.4 Warnings

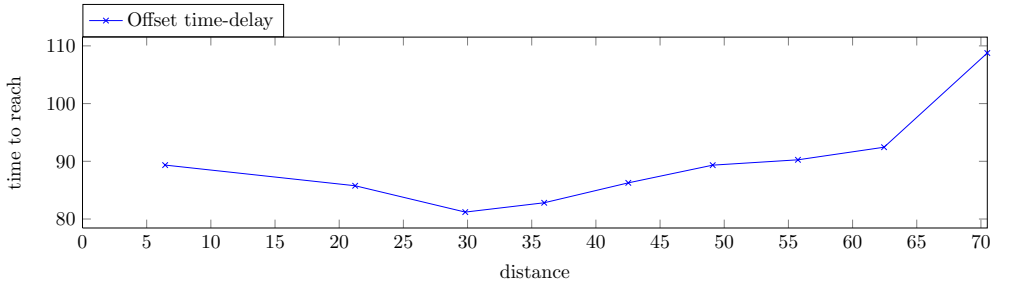
1 Z1 COMPACT-2014-9-3-16:31:10-400mms.mkv

1.1 Information

...

1.2 Empirical Result





1.3 Data

Average display update time: 18.1667±0.897527ms
Average display update step: 6.868±0.78536mm
Robot actual speed: 400mm/s
Calculated average robot speed: 393.637±0.649598mm/s
Calculated average object speed: 393.965±4.40516mm/s
The robot starts moving at time 1460.79 and it takes 91.2069 ms for the screen to respond
Median time delay: 89.3334
Median distance offset: 25.8276

1.4 Warnings

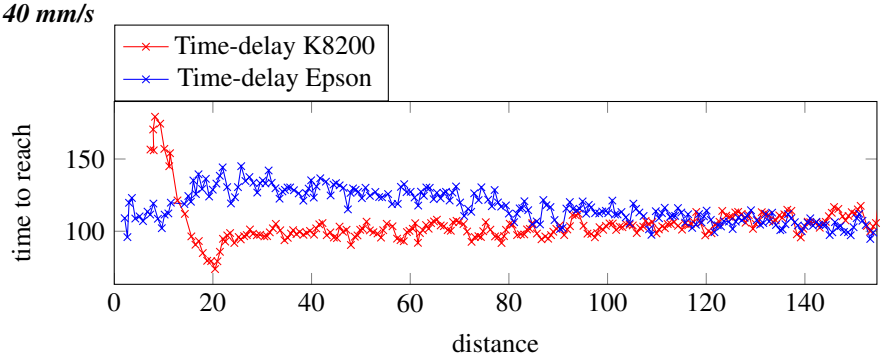
- WARNING 4: Time-steps found differs more than 10% from theoretical amount of time-steps. Diff = -28.200001%
- WARNING 8: Suscpected framedrop(s) detected.

A.2 Merged results different velocities

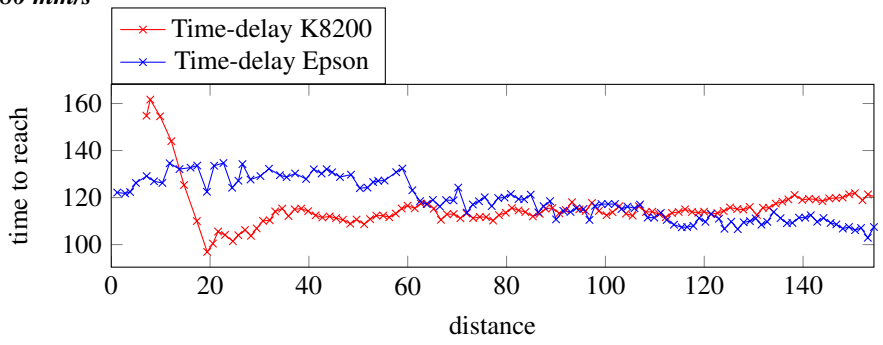
The following plots shows the time-delay plot from each product and velocity merged for easier comparison.

Tablet comparison

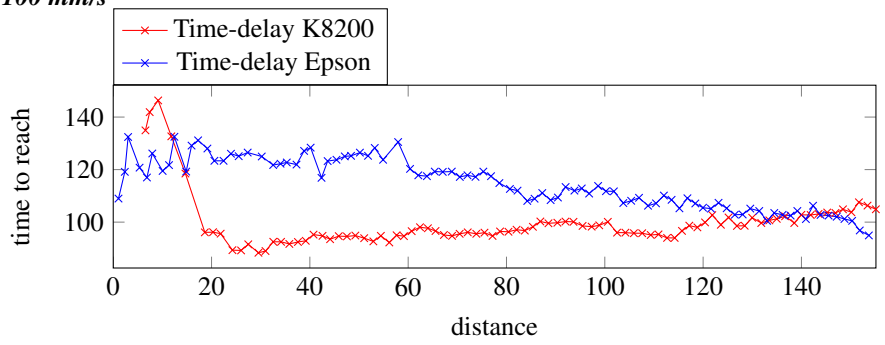
40 mm/s



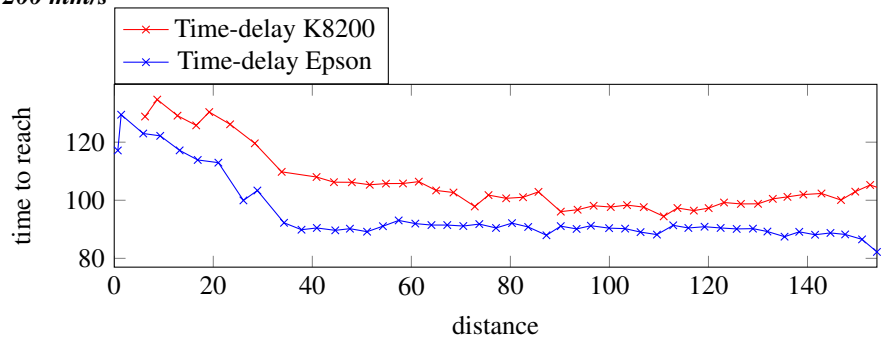
80 mm/s



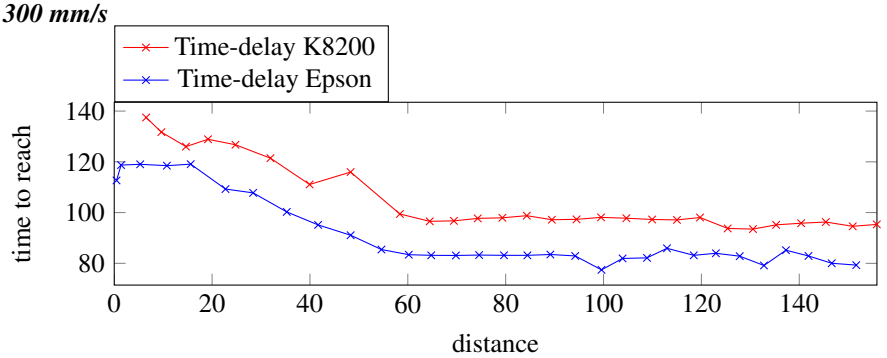
100 mm/s



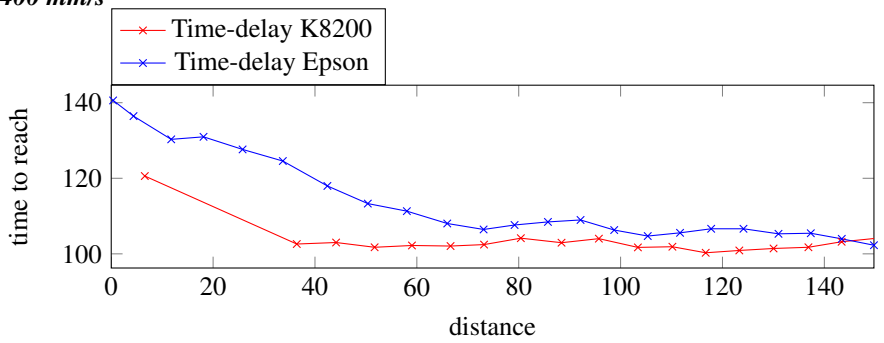
200 mm/s



300 mm/s

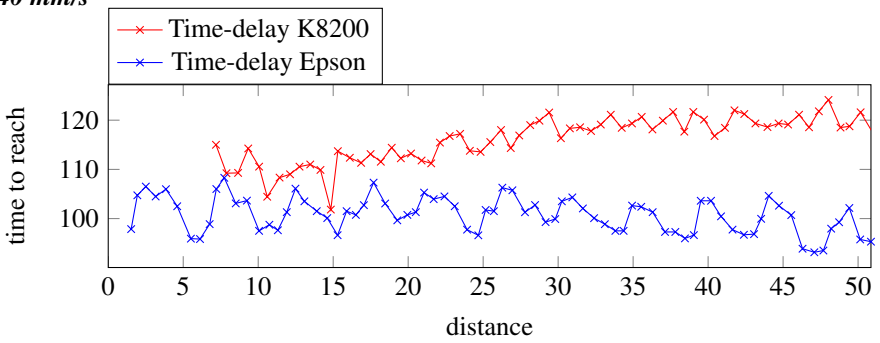


400 mm/s

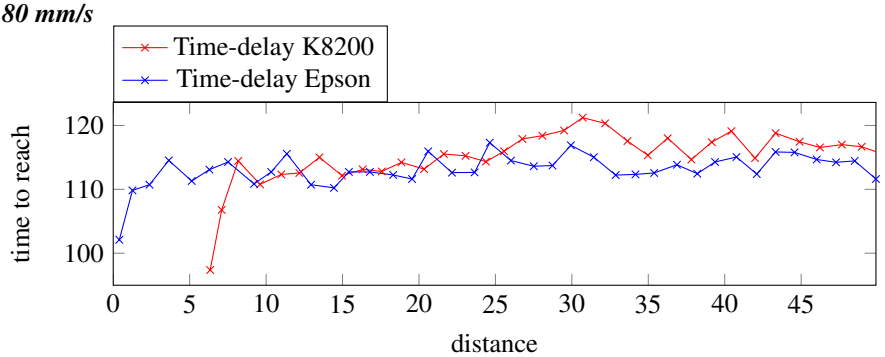


Small Phone

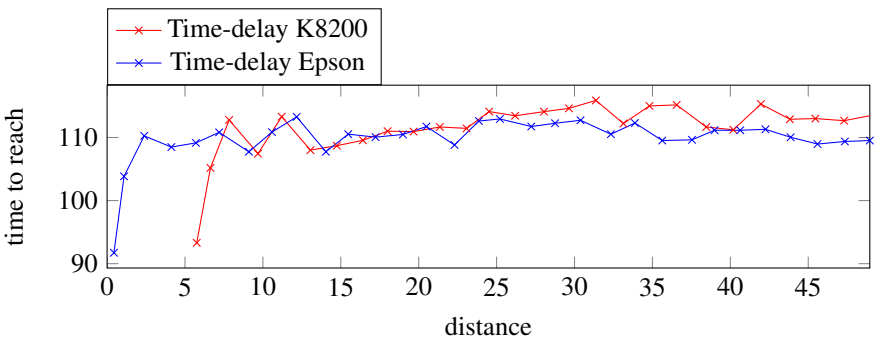
40 mm/s



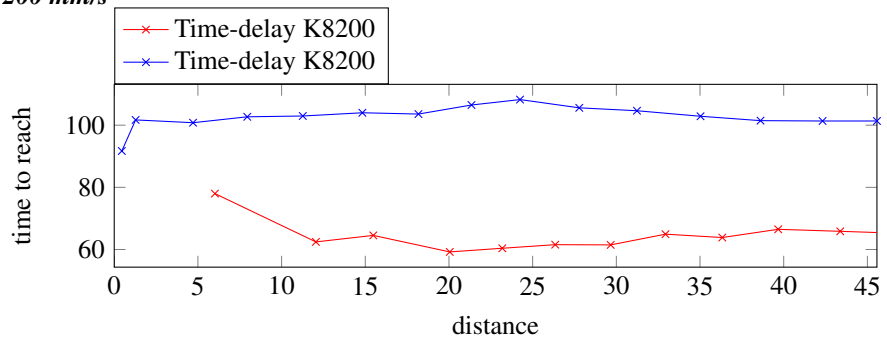
80 mm/s



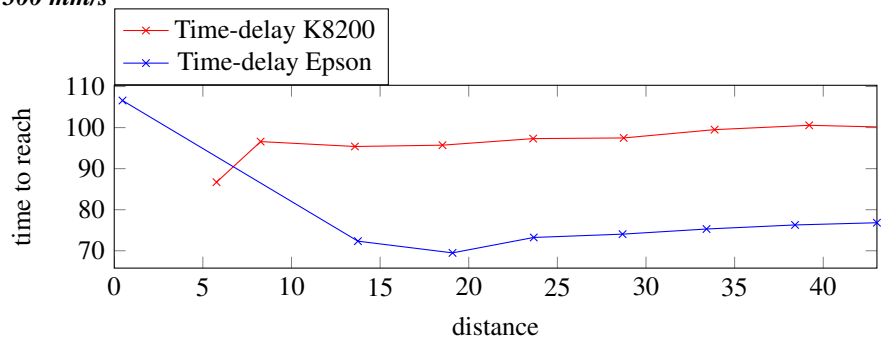
100 mm/s



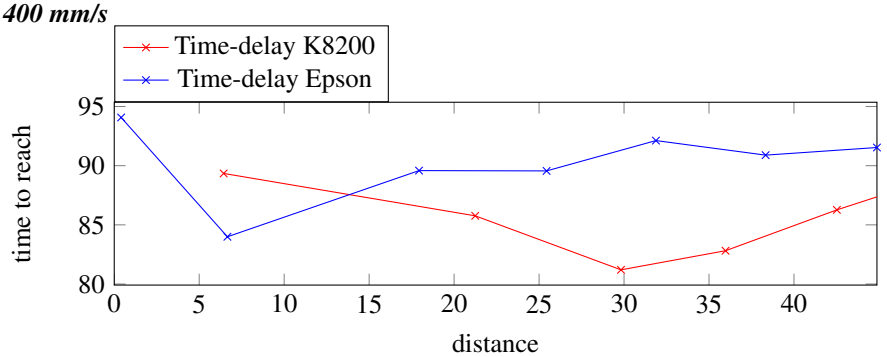
200 mm/s



300 mm/s



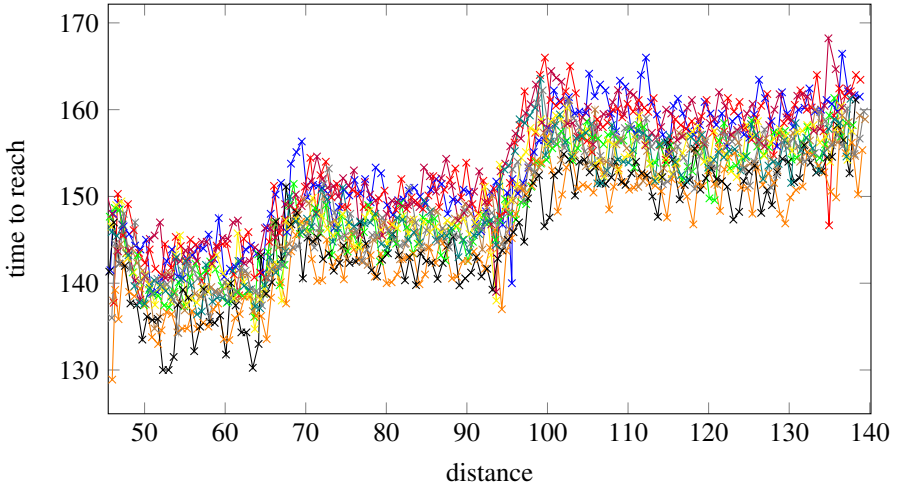
400 mm/s



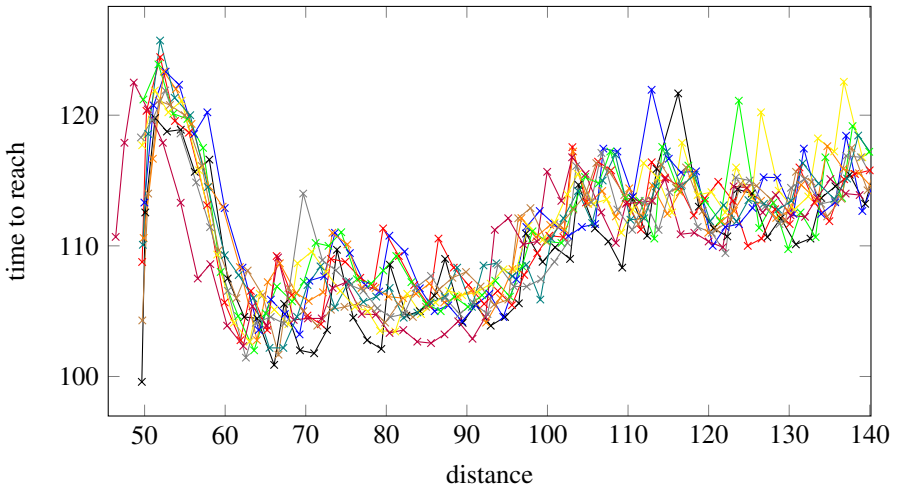
A.3 Multiple swipes, same velocity

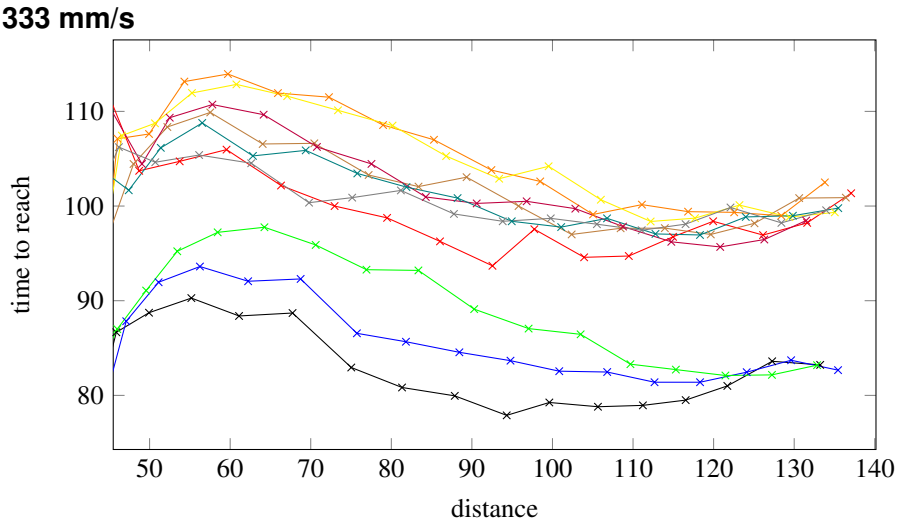
Each of the following plots shows data from 10 swipes performed on the same product and velocity. These test are not performed on any of the products shown earlier in this appendix.

40 mm/s



100 mm/s




333 mm/s
time to reach
110
100
90
80
50
60
70
80
90
100
110
120
130
140
distance

Bibliography

Android, DisplayMetrics (Accessed: 2015-03-25). Google. URL: http://developer.android.com/reference/android/util/DisplayMetrics.html.

Annheimautomation (Accessed: 2015-03-25). URL: http://www.anaheimautomation.com/manuals/forms/stepper-motor-guide.php.

Arcam AB (Accessed: 2015-03-25). Arcam AB. URL: http://www.arcam.com/technology/electron-beam-melting/.

Arduino (Accessed: 2015-03-25). Arduino. URL: http://www.arduino.cc/.

Blom, G., J. Enger, G. Englund, J. Grandell, and L. Holst (2010). *Sannolikhetsteori och statistikteori med tillämpningar*. Studentlitteratur.

Createing Bounding boxes and circles for contours (Accessed: 2015-03-25). Itzeez. URL: http://docs.opencv.org/doc/tutorials/imgproc/shapedescriptors/bounding_rects_circles/bounding_rects_circles.html.

Freedman, D. H. (2011). *Layer by Layer*. Technology Review. URL: http://www.technologyreview.com/featuredstory/426391/layer-by-layer/.

Goldt, S., S. van der Meer, S. Burkett, and M. Welsh (1995). "The linux programmer's guide". URL: http://www.tldp.org/LDP/lpg/node15.html.

Harris, W. (Accessed: 2015-03-25). *How 3-D Bioprinting Works*. HowStuffWorks. URL: http://health.howstuffworks.com/medicine/modern-technology/3-d-bioprinting1.htm.

Kickstarter (Accessed: 2015-03-25). Kickstarter, Inc. URL: https://www.kickstarter.com.

Marlin (Accessed: 2015-03-25). URL: http://reprap.org/wiki/Marlin.

National Inventors Hall of Fame (Accessed: 2015-03-25). URL: http://invent.org/inductees/hull-charles/.

OpenCV (Accessed: 2015-03-25). OpenCV Documentation. Itseez. URL: http://opencv.org.

Peachy Printer (Accessed: 2015-03-25). URL: http://www.peachyprinter.com/.

PGFPlots - A LaTeX package to create plots (Accessed: 2015-03-25). URL: http://pgfplots.sourceforge.net/.

reprap.org (Accessed: 2015-03-25). URL: http://reprap.org.

Sweet, M. R. (2010). "Serial programming guide for posix operating systems". URL: http://www.tldp.org/LDP/lpg/node15.html.

Velleman (Accessed: 2015-03-25). Velleman 3D-printer. URL: http://www.k8200.eu/.

Ximea (Accessed: 2015-03-25). Technical specification - MQ003CG-CM. Ximea. URL: http://www.ximea.com/en/usb3-vision-camera/xiq.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER´S THESIS

Date of issue
March 2015

Document Number
ISRN LUTFD2/TFRT--5967--SE

Author(s)
Ragnar Wernersson

Supervisor
Magnus Midholt, Sony Mobile
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden
Rolf Johansson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Sponsoring organization

Title and subtitle

Robot Control and Computer Vision for Automated Test System on Touch Display Products

Abstract

The goal of this master thesis is to set up a low cost automated robotic test system which can later be reproduced to greatly increase test coverage. Mostly through experimental research this thesis will find good components to use and evaluate these from a performance and cost perspective. It will also develop computer vision algorithms needed and automation software for the final set up. The performance of the robotics will be investigated by comparing with an industrial robot.

A modified 6000 SEK 3D-printer was selected and proved to work. The computer vision was developed using OpenCV and a fully automated system with appropriate resulting plots was created. Comparing with the industrial robot the setup using a 3D-printer proved to work better because it allowed for better positioning of the camera. It was also concluded that the selected robot system based on a 3D-printer was capable enough and would drastically lower the space and cost from a system using an industrial robot.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-117

Recipient's notes

Security classification