|  |  |  |  |
| --- | --- | --- | --- |
| Decorative | | | |
|  |  |  |  |
| PID: AUTONOMOUS CONTROL  **Year 13 Computer science project**  **LUKE WALTON** | |

Contents

[Requirements Analysis 5](#_Toc130555756)

[Introduction of project/scope 5](#_Toc130555757)

[Identification of problem 5](#_Toc130555758)

[Identification of the prospective user(s) 6](#_Toc130555759)

[Interview with Sponsor 6](#_Toc130555760)

[Sponsor Interview – Mr Maher 7](#_Toc130555761)

[Research into current or alternate methods 8](#_Toc130555762)

[PowerPoints 8](#_Toc130555763)

[Worksheets 8](#_Toc130555764)

[OneNote Work 9](#_Toc130555765)

[Interview end users 9](#_Toc130555766)

[End User Interview 1 – Alex E 9](#_Toc130555767)

[End User Interview 2 – Jakub W 10](#_Toc130555768)

[User Needs 11](#_Toc130555769)

[Requirements and Limitations 12](#_Toc130555770)

[Requirements 12](#_Toc130555771)

[Limitations 12](#_Toc130555772)

[Objectives (SMART) 13](#_Toc130555773)

[Technical Analysis 14](#_Toc130555774)

[Technology comparisons 14](#_Toc130555775)

[Objective 14](#_Toc130555776)

[Python 14](#_Toc130555777)

[C# 14](#_Toc130555778)

[Scratch 14](#_Toc130555779)

[Chosen Technology 16](#_Toc130555780)

[Python 16](#_Toc130555781)

[C# 16](#_Toc130555782)

[Scratch 16](#_Toc130555783)

[Chosen Technology 17](#_Toc130555784)

[Architecture overview 17](#_Toc130555785)

[Mathematical model 18](#_Toc130555786)

[Proportional 18](#_Toc130555787)

[Integral 19](#_Toc130555788)

[Derivative 20](#_Toc130555789)

[Output 21](#_Toc130555790)

[Gap Analysis 22](#_Toc130555791)

[Prototype 23](#_Toc130555792)

[MainWindow.xaml 23](#_Toc130555793)

[MainWindow.Xaml.Cs 23](#_Toc130555794)

[NavMenu.Xaml 23](#_Toc130555795)

[NavMenu.Xaml.Cs 24](#_Toc130555796)

[Model1.Xaml 25](#_Toc130555797)

[Model1.Xaml.Cs 26](#_Toc130555798)

[Model 2 and 3 26](#_Toc130555799)

[Summary 26](#_Toc130555800)

[Functional Design 28](#_Toc130555801)

[UI Design 28](#_Toc130555802)

[UI Sketches 28](#_Toc130555803)

[Full Design 29](#_Toc130555804)

[Hierarchy Diagram 30](#_Toc130555805)

[Data model algorithms 31](#_Toc130555806)

[Technical Design 32](#_Toc130555807)

[UML 32](#_Toc130555808)

[Class Breakdown 33](#_Toc130555809)

[Preamble 33](#_Toc130555810)

[Generic Model: 33](#_Toc130555811)

[NavMenu 34](#_Toc130555812)

[PID 34](#_Toc130555813)

[Model 1 – Rotation: 35](#_Toc130555814)

[Model 2 – Position: 36](#_Toc130555815)

[Model 3 – Velocity: 38](#_Toc130555816)

[Technical Solution 40](#_Toc130555817)

[PID.PID.cs 41](#_Toc130555818)

[GenericModel.cs 42](#_Toc130555819)

[MainWindow.xaml 43](#_Toc130555820)

[MainWindow.xaml.cs 43](#_Toc130555821)

[NavMenu.xaml 43](#_Toc130555822)

[NavMenu.xaml.cs 45](#_Toc130555823)

[RotationModel.xaml 46](#_Toc130555824)

[RotationModel.xaml.cs 47](#_Toc130555825)

[PositionModel.xaml 50](#_Toc130555826)

[PositionModel.xaml.cs 52](#_Toc130555827)

[VelocityModel.xaml 57](#_Toc130555828)

[VelocityModel.xaml.cs 59](#_Toc130555829)

[Systems Testing 64](#_Toc130555830)

[Testing Against Objectives 64](#_Toc130555831)

[Testing Handling Exceptions 66](#_Toc130555832)

[Evaluation 69](#_Toc130555833)

[Post Production Interviews 69](#_Toc130555834)

[Sponsor Interview 69](#_Toc130555835)

[End-User Interviews 69](#_Toc130555836)

[Summary 71](#_Toc130555837)

# Requirements Analysis

## Introduction of project/scope

PID is a closed feedback loop using Proportional, Integral, and Derivative calculation, designed to meet a numerical goal without overshooting. This is often used in automatic systems, for high speed and accuracy.

Some terms that I will be using and are important to understand for this are:

* Set Point – The desired numerical value to be reached
* Process Value – The current numerical value, read in by a sensor
* Error – The difference between the Set Point and Process Value
* Output – The calculated PID output, often an acceleration
* kP, kI, kD – The constants used in each individual calculation

I intend to build a teaching tool able of demonstrating mathematical principles, like Integration and Derivation, in a fun and interesting way for the students to see. I think to be able to see calculations being ran and used in real time will really help them develop an interest in what’s happening behind the screen, boosting their focus, and hopefully helping them remember and recall what they are learning.

Alternately, the PID module could be separated and used for robotics, such as autonomous control systems. I’ll get onto more examples of this later. The PID algorithm itself is usable in any given context, that can be broken down to a numerical goal, and acceleration, provided it is tuned correctly via the passed in parameters, like P, I and D constants, or clamping and scaling values.

## Identification of problem

Imagine a thermostat. When you set it to a particular temperature, you want it to reach that exact point quickly and accurately. Behind the screen there will be a PID module controlling the heating, turning it up and down perfectly so that it reaches the set temperature, without overshooting. If the thermostat fully applied the heating until it reached the correct temperature, and then turns off, the room will continue heating up for a bit afterwards still, as the signal to turn off will not happen instantly. And then after this, it will only slowly cool down again, so some extra control is necessary, including shutting off before the point is reached.

This is where PID comes in. There’ll be a PID model reading the Process Value from the temperature sensor and giving its output to control the heating. Now imagine those exact same principles but controlling a servo or piston to reach a distance or angle, and now you have a robotic arm, picking up a can of cat food off a conveyor belt, sorting it, and putting it onto another belt or box.

## Identification of the prospective user(s)

Usable in maths for demonstrating the uses of Proportional, Integral and Derivative relationships, Maths teachers and students alike could use and benefit from this tool, being able to see the process that must be done, then watching how the calculations work, what they do for the system, and being able to influence it themselves.

However, it is also applicable to Mechanics in Maths and Physics as the models demonstrate a physical autonomous system moving and compensating for momentum and air resistance. The models representing these systems - keeping a rocket upright, or automatically driving a car - would be very helpful to study how the system compensates for these outside factors that the students will be studying.

When put into a program, it becomes open to Computer Science students and teachers, demonstrating autonomous systems and feedback loops. The PID module itself being useful in robotics, which has strong ties to computer science, but also the programming principles demonstrated by using the one module in multiple front-end models.

## Interview with Sponsor

The purpose of an interview with the sponsor is to determine the [User Needs](#_User_Needs), which will be used to establish [Requirements and Limitations](#_Requirements_and_Limitations) and then [SMART Objectives](#_Objectives_(SMART)) to aim for throughout production.

|  |  |
| --- | --- |
| **Question** | **Purpose** |
| What teaching methods are currently used for teaching Maths, Physics and Computer Science? | The point of this question is to learn more about the current options that can be replaced by this project. |
| How engaged with the lesson do you find students using these teaching methods? | This question helps us know more about the current teaching methods, to see what needs changing more |
| Do you think having an interactive tool demonstrating the taught principles would help? | This gets a teachers opinion on how much this tool will help with engagement and recollection. |
| What do you want to get out of the program? | This and the next one help form the [User Needs](#_User_Needs) and [Requirements And Limitations](#_Requirements_and_Limitations) |
| What limits will you have for running it? |  |

## Sponsor Interview – Mr Maher

**Me:** What teaching methods are currently used for teaching Maths, Physics and Computer Science?

***Mr Maher:***Often I will put a PowerPoint up on the board, talking through it and answering anyone’s questions. Later I’d provide worksheets for the students to try, and occasionally put a video on the board.

**Me:** How engaged with the lesson do you find students using these teaching methods?

***Mr Maher:*** Often students will ask a lot of questions that either I or the PowerPoint have already gone over, so they clearly either don’t take in the information, or they stop paying attention. I’d say overall there’s about 5-10% of students not paying attention in a lesson.

**Me:** Do you think having an interactive tool demonstrating the taught principles would help?

***Mr Maher:*** Absolutely, just a video alone helps grab their attention, and often the taught principles stay in their mind better, so a program that they can interact with and see how it reacts would likely help a lot.

**Me:** What do you want to get out of the program?

***Mr Maher:*** I want my students to be able to interact with the program, see how that interaction is affecting it, and see what is happening in the background. Most importantly I want it to hold their attention, but still actually teach them.

**Me:** What limits will you have for running it?

***Mr Maher:*** It will likely be running on school computers or laptops, so it can’t require extremely heavy processing power. Ideally, we’d also have it ready for next year’s students, or even by this year’s revision period.

In conclusion, the program should hold the user’s attention, so that they aren’t bored, but not to a level they just aren’t learning. It should be interactive to properly demonstrate the principles being used, be quick and light to run so that it will be able to run on school laptops and computers and be ready for this year’s revision period.

## Research into current or alternate methods

### PowerPoints

All three subjects are currently taught by putting a PowerPoint on the board, and the teacher talking through it. This is often boring, as it will just be a block of text or a single image for either the student or teacher to read and will lead to the students stopping paying attention and not learning what is being taught.

PowerPoints can be helpful for demonstrating some things in diagrams or small animations, and often these are projected onto white boards, so the teacher can easily annotate it, describing processes in more detail if required.

This could couple well with my project, as the teacher can teach the principles and then demonstrate them easily by either projecting the program, or having students run it and interact with it themselves. If projected, the teacher will be able to control the process, demonstrating the processes in the best way possible, before letting the class get on with the program themselves, using their curiosity to fuel their learning.

### Worksheets

Worksheets are good checkpoint tests to see if a student understands what they’ve been taught, however if they don’t understand it, often the worksheet does nothing to help them learn what they need to know. They are forced to ask either another student or the teacher for help. This often leads to many students not learning, as they are too afraid to ask for help, as they think it will reflect a poor image onto them.

Due to this last point, the project would work well with this method, as they would not need to ask another person for help, they can simply run the program and see what it is doing, and hopefully learn from it. This cuts out the fear of being judged, even if they likely wouldn’t be in the classroom environment anyway.

### OneNote Work

Using OneNote, the teacher is able to send out pages of work and the lesson PowerPoints to the students. This means that it is a lot easier for students to individually take notes or look back at the slides while doing the work, as they can do this themselves rather than getting the teacher to do it and stalling the entire class.

This will also work well combined with the program, as it can be put onto a page and distributed so that the students can run it on their devices, and interact with it themselves, once again rather than every student interfacing with it through the teacher projecting it once onto the screen.

## Interview end users

The purpose of an interview with end users is to get the flaws of the current system and how we can increase the student’s engagement from the students’ point of view

|  |  |
| --- | --- |
| **Question** | **Purpose** |
| How easily did you learn Differentiation and Integration? | Identify the students’ ability with standard methods, how much could they have improved with this project? |
| Did you find yourself engaged with the lessons? | See if they were engaged from their point of view, was it possible for them to me more engaged? |
| Do you think an interactive tool, demonstrating these principles and uses, would help with engagement? | This gets the students’ opinion on how much this tool will help with engagement |
| Do you think said tool would help with remembering the methods? | This gets the students’ opinion on how much this tool will help with recollection |
| What kind of computer system do you have at home, if any? OS and power wise | This helps establish the [Limitations](#_Requirements_and_Limitations) of the project, as it will have to run on every students devices. |

### End User Interview 1 – Alex E

**Me:** How easily did you learn Differentiation and Integration?

***Alex:*** I learnt it just fine, it made logical sense and after some worksheets, it stuck well in my mind.

**Me:** Did you find yourself engaged with the lessons?

***Alex:*** No, they were often very boring, with a lot of reiterating the same thing that I learnt quickly. After a weeks’ worth of worksheets, more worksheets don’t help anymore.

**Me:** Do you think an interactive tool, demonstrating these principles and uses, would help with engagement?

***Alex:*** Massively. Any interactive tool that I can play around with, and see how it reacts, especially with feedback of what is happening inside, would deeply benefit my learning.

**Me:** Do you think said tool would help with remembering the methods?

***Alex:*** Same as above, I will more easily recall something if I enjoyed learning it, or an experience closely tied to something, as it will be more stuck within my memory. I feel an interactive tool would massively help.

**Me:** What kind of computer system do you have at home, if any? OS and power wise

***Alex:*** I have a moderately powerful gaming PC, running windows 11 currently.

From Alex’s point of view, it isn’t a necessary tool as he was fairly strong anyway, however it would strongly benefit his attention to the lesson and possibly help him catch something he might have missed. There is not a low barrier for computer power, for Alex, but he is a perfect example of why this should be done by the start of the next school year.

### End User Interview 2 – Jakub W

**Me:** How easily did you learn Differentiation and Integration?

***Jakub:*** I was quick at understanding what each one did, like finding area or gradient, but it took me a lot longer to understand how to do it myself. I found myself often mixing up which one to add or subtract the power, or multiply or divide by etc.

**Me:** Did you find yourself engaged with the lessons?

***Jakub:*** Maybe the first lesson on each, but after that it was just many worksheets, trying to figure out what to do for each as I went through them. This meant I was often losing focus, not having the mental effort to push over this hurdle.

**Me:** Do you think an interactive tool, demonstrating these principles and uses, would help with engagement?

***Jakub:*** Probably yes, it would be a lot less boring than just working through worksheets, yet I probably wouldn’t just mess around with the tool, I’d actively try to learn.

**Me:** Do you think said tool would help with remembering the methods?

***Jakub:*** It would depend on how well the program demonstrates it. If it was just hidden in the code and I had to scour through to find it, then no, but if there was an easily accessible explanation and demonstration, then it would help a lot, it would be a lot more memorable than text on a whiteboard.

**Me:** What kind of computer system do you have at home, if any? OS and power wise

***Jakub:*** I do have a computer at home, but not very powerful. Windows Operating system.

From Jakub’s point of view, the program would have heavily benefitted getting the method stuck into his head, seeing it in action whenever he needs. The program will have to be light to suit to Jakub’s use and must demonstrate it well in the program and UI, rather than running something and trying to figure out what it is doing in the background.

## User Needs

After all the interviews, it seems clear that everyone thinks that this program would really help engagement and recall of information. I do need to ensure that it isn’t a program that can just be played around with endlessly, meaning actual work won’t be done.

I need to ensure the program is light to run, as although Alex has a stronger device, Jakub doesn’t, and the school provided devices aren’t strong. So, people should be able to run it on school devices and weaker home computers. If it can’t be easily run on these devices, it leaves only people with their own stronger devices able to run it, at which point I am limiting the opportunities of people with disadvantaged backgrounds, which I don’t want, so it must be easy to run.

Although none of the interviews brought up accessibility requirements, my program has no apparent need for a keyboard control, only a mouse or touchscreen would actually be required. This will be a good goal to aim for to help not restrict the possible users, as outlined in the previous paragraph.

The program must give a lot of feedback clearly, without the need to look through the code. It should be easily accessible and give a wide range of information on what is actually happening, including equations for the Derivative and Integral, as well as feedback on the current values and what is currently happening in the program. This is so it can actually be learnt from.

Finally, it wants to be finished by this year’s exam period, so that the models can be used for revision. Although the tool will be a lot more useful for initial learning rather than revision, it will still hopefully help, and Spring 2023 is still a good goal to have it ready for next year’s students learning differentiation and integration for the first time.

So in summary, my user needs are:

* Interactive and Engaging, but not “Fun”
* Light on the system
* Clear feedback of the process
* Finished by spring of 2023

## Requirements and Limitations

Using these user needs, I am able to outline the requirements and limitations for the project. This should encompass all three interviews and their needs. It should also start to incorporate programming principles of how some of these can be achieved or what restrictions will do for the program.

### Requirements

* Able to hold the user’s attention
  + Interactive
  + Interesting range of scenarios
* Helps cement the information into the user’s head.
  + Information on the process easily available
  + A demonstration where you can see what is happening as it goes.
* Follows laws of physics
  + Air resistance and momentum modelled.

### Limitations

* Finished before this year’s revision period.
  + Spring 2023
  + If not possible, aim for September 2023   
    for next year’s students
* Simple and light to run.
  + O(nk) at worst
  + Simple, light graphics
  + Physics abstracted to simpler degrees.
  + Proportional relationships

## Objectives (SMART)

Using these [requirements and limitations](#_Requirements_and_Limitations), we can form a list of objectives to aim towards throughout production. These should follow SMART principles (Specific, measurable, achievable, realistic, time based) to ensure the programming process can go quickly while still delivering the product that the sponsor and end users need.

1. Interactive and intuitive user interface (A, R)
2. Contains a control panel to interact with the PID model (S,M,A,R)
3. Includes feedback of calculations and current values (M,A,R)
4. Has expandable menus detailing the processes behind each part of PID (S,M,A,R)
5. Three detailed models should be available (M,A,R)
6. Rotation, extension, and velocity control should be demonstrated (S,M,A,R)
7. Relationships should aim to be proportional or polynomial (A,R)
8. Must be light enough to run on light computer systems, running windows (S,M,A,R)
9. Must be finished before Spring exams 2023 (S,M,A,R,T)

# Technical Analysis

## Technology comparisons

Using these Objectives, I can now start to compare different programming languages, and tools within the language, to each other. This helps me decide which language I should use, and which would be definitively best for the program.

|  |  |  |  |
| --- | --- | --- | --- |
| Objective | Python | C# | Scratch |
| Interactive and intuitive user interface (A, R) | TKInter is a useful tool for UI design in python, however it is primitive for rotating and changing objects | WPF is extremely helpful for this project, as it has built in functionality for changing length, position, or rotation of an object | Scratch is simple and has inbuilt functionality for animating sprites, however it is primitive in its design due to its simplicity |
| Contains a control panel to interact with the PID model (S, M,A,R) | TKInter seems to have functionality for interactive controls, however it is unintuitive and difficult to learn, from my research | WPF has inbuilt buttons, as well as an easy Function menu to create On Click Events. | Scratch has simple When Key Pressed or When Sprite Clicked blocks, so could make a UI with ease |
| Includes feedback of calculations and current values (M, A,R) | Python has a simple console Print() function, and Tkinter has a Text item, and so feedback can be easily returned as text. There is also a helpful library called MatPlotLib for plotting graphs. | With sliders, colour control and text blocks, WPF is a great option for returning feedback | Scratch has limited text return options, and so would not be a good choice. |
| Has expandable menus detailing the processes behind each part of PID (S,M,A,R) | TKInter has dropdown menus, however they look old and unappealing visually. This would not help hold the user’s attention | While there is the ability to create custom drop-down menus, it isn’t entirely intuitive, so WPF isn’t the best choice here, but it works. | Scratch has changeable textures on sprites. With these texture changes being able to be attached to any trigger, Scratch is especially useful for changeable, expandable menus |

|  |  |  |  |
| --- | --- | --- | --- |
| Three detailed models should be available (M, A,R) | I could use Functions to store different models and navigate between them through a menu. Returning after the PID module won’t be difficult either. | C# uses Object Orientated Programming (OOP) so having different models call one PID module will be simple | Scratch will easily be able to make 3 different models, and run a different one based on a menu selection, but the PID model would mean the structure ended up messy quickly |
| Rotation, extension, and velocity control should be demonstrated (S,M,A,R) | Python does not easily handle rotation, extension, or position change in TKInter | All three are really easy and simple to implement with WPF | Scratch has really simple blocks for movement, rotation, and extension. It also has variable functionality to store velocities |
| Relationships should aim to be proportional or polynomial (A,R) | As it’s a text-based programming language, the maths will be as simple as I make it. | As it’s a text-based programming language, the maths will be as simple as I make it. | Scratch has a variety of maths blocks, however due to the block nature it will actually be a lot slower to run a small and simple algorithm. |
| Must be light enough to run on light computer systems, running windows (S,M,A,R) | Due to the difficult nature of changing TKInter UI, the program will be more complex than it has to be, slowing it all down. It also requires a python interpreter to run on windows, which creates unnecessary complexity | Although a very easy language to compress the algorithms into simpler statements, WPF can thoroughly slow down if too many updates are required. Fortunately, this shouldn’t be an issue for the three models. | Scratch would be entirely illogical to use, as it requires an internet connection to access the site and encourages students to browse games made available on the site. |
| Must be finished before Spring exams 2023 (S,M,A,R,T) | I already know how to program in Python, however some functions in TKInter may take some time for me to learn and use efficiently. | I am near fluent in C# and WPF, so I will be able to produce the program very quickly without a need for much more research or sacrificed quality of the product. | Being a block-based language, it would take a ridiculous time to code the complex algorithms into scratch, and so wouldn’t be a fit choice. |

## Chosen Technology

### Python

I am already well versed in Python, but not so much TKInter or MatPlotLib. This would mean the program would be slow to develop as I am spending time learning the ins and outs of these libraries. From the research I’ve done so far on these libraries, the tools they contain are simply not suited for the required needs of this project – the functions either don’t exist, are sloppy and difficult to use, or they are not visually appealing. When it comes to actual algorithms and such, Python would work as well as any text-based programming language, with most of the same functions for what I need. Overall, Python wouldn’t be a good choice, but it would be doable provided there wasn’t a better option available for me to use for the project.

### C#

With WPF and OOP, C# is an extremely reliable choice for a program. I already know the language well, including the extra libraries I might need like WPF, and these libraries give me extensive control over the UI, feedback to the user and ensuring it is all visually appealing throughout. OOP also opens the door for an interesting structure to allow the connection between all three models and a single PID controller module, that they could all call in different contexts, and get the result they want from it. With a wide diversity of methods available, it is quickly apparent than C# is an outstanding choice for this project, and most likely the best one.

### Scratch

While Scratch is an amazing tool for people first learning programming, or to throw a quick simple demonstrative tool together – it isn’t a good choice at all for something that requires complicated code. Its UI and feedback system is extremely simple, and amazingly controllable, however due to this simplicity and its limited functionality or blocks available, the moment you want to code a serious program with heavy algorithms, it simply doesn’t stand up to other programming languages. In conclusion, Scratch is a fantastic UI tool, and likely better for it than TKInter in python (which really says something about TKInters quality, that it can be beaten by a toy for children) but when it comes to the actual functionality and coding aspect, it falls flat on its face.

### Chosen Technology

After reviewing each Language, C# with WPF is clearly the best choice for this program, with Scratch and Python barely holding a candle to its ability to provide what I need for the program. Scratch and C# both provide the intuitive and appealing UI, while Python and C# hold the algorithmic diversity I need, and so to have the best of both worlds, C# is the clear choice for this project.

## Architecture overview

My UI will consist of three models navigable between through a small heading. They will be controlled via a panel with sliders to control the kP, kI and kD, and a feedback system. These will be required for all three models, and so will be persistent.

The first model will use PID to control the rotation of a rectangle to maintain an upright position. The user will be able to adjust the rotational velocity of the rectangle, before enabling PID with their selected values for the constants. The model will involve air resistance, and the Output will be linked to Rotational Acceleration, so that the automatic system will try reach an upright position, without overshooting.

The second model will still control rotation; However, it will be in three different points, and two of them will actually be controlling a distance. This is because the model will be of a Robotic Arm able to move to any point within a circular range. One of the rotations will be controlling the base, deciding what direction the base of the arm is pointing. The next rotation is controlling the angle between the horizontal and the first length of the arm. The final rotation control is between the first and second length of the arm, acting as the elbow of the arm. These last two will form a triangle between the two lengths of the arm, and the level of the floor modelled. This will not account for vertical movement, so the desired value for these two can be calculated using trigonometry, and passed into the SP.

The third model will control both rotational velocity, and standard velocity. The rotational model will be used on the front wheels of a vehicle, and the velocity model will be used to control the engine and thus velocity of the vehicle. This effectively creates a self-driving car, that can navigate to a point. However, there will be no obstacles or other cars that must be avoided; It is only the very core functionality of an automatic car system. Similar to the previous model, the user will select a point on a canvas that they want the vehicle to navigate to, and the PID system will do the rest.

## Mathematical model

At the core of all three models will be the PID controller. I spoke briefly on this earlier, so you already know a bit of how this works.

Here are the definitions for some of the important terms again:

* Set Point – The desired numerical value to be reached
* Process Value – The current numerical value, read in by a sensor
* Error – The difference between the Set Point and Process Value
* Output – The calculated PID output, often an acceleration
* kP, kI, kD – The constants used in each individual calculation

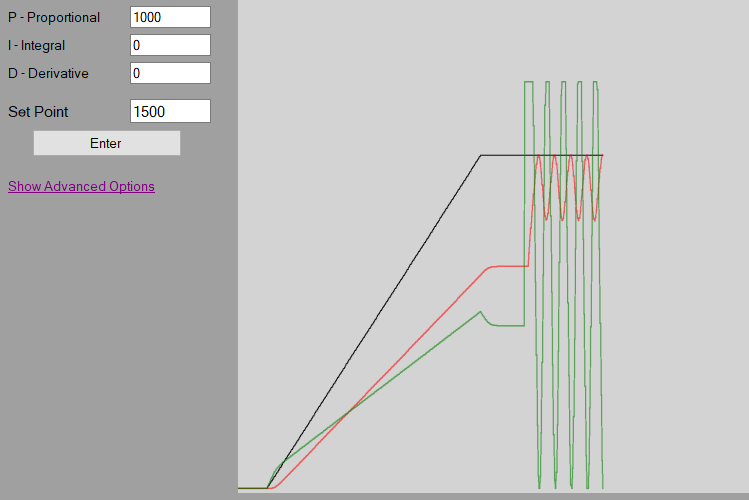
The controller can be broken down into four calculations: Proportional, Integral, Derivative and Output. All of these can be nicely demonstrated as graphs between the Set Point (Black), Process Value(Red) and the Output (Green), with time across the X axis and value on the Y axis.

### Proportional

The proportional calculation forms a linear, proportional relationship between the Error and the output. This is done just by multiplying by the kP constant, as such:

P = Err \* kP

This creates a relationship that simply approaches the target with constant acceleration, so that at an error of zero, there is no acceleration. However, because of momentum, the process value will not stop exactly when the output is zero, it will keep going at the highest speed it reached, and only then will it start to decelerate and turn around. This causes an entropic loop of oscillating back and forth, when used alone.



https://pidexplained.com/pid-controller-explained/

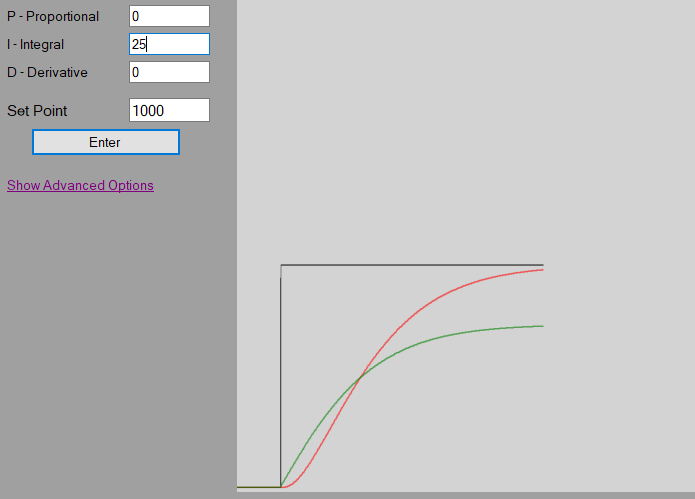
### Integral

The integral calculation creates a relationship of slow increasing acceleration. It will do fairly little at the beginning, but as it takes more and more time to reach the Set Point, the Output will slowly increase by more and more each time. This means that the longer it takes to reach the set point, the faster and faster it will increase acceleration (Jerk increases)

I = kI \* Err \* Δt

It = It + I

These calculations come together to form this relationship through the accumulative total, added to every time the PID controller is called. This would increase really quickly and uncontrollably, however in the initial calculation there is a delta time, so regardless of the time period between calculations, it will increase at the same rate. Over two seconds, whether the PID controller is called one time, or ten, the integral total will still be the same at the end of the two seconds. Due to the increase over time, integral becomes useful for counteracting resistive forces.



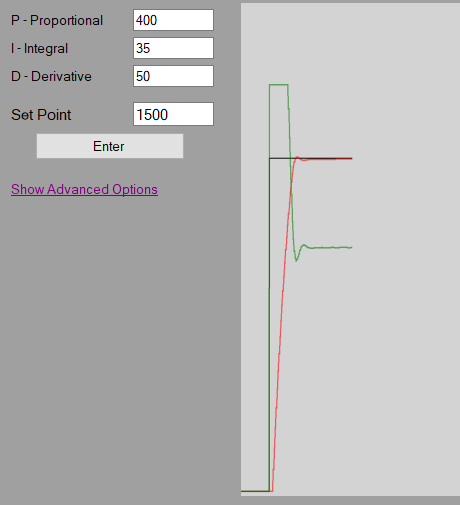
https://pidexplained.com/pid-controller-explained/

### Derivative

The Derivative calculation is actually the only relationship that turns negative before reaching the set point. This means that on its own, it is actually terrible at reaching the set point, it will just quickly and smoothly come to a stop. Only once we introduce an amount of Proportional or Integral will it actually start to reach the goal.

D = kd \* (Err – pErr) / Δt

The derivative works in the same way as the integral calculation, as it has delta time involved in the calculations, to help account for the time taken between the previous PID calculation, and this one. This is required in derivatives as the calculation involves subtracting the previous error from this one. This means if the error is reducing, because the PV is approaching the SP, then D will be negative, and it will slow down to a stop. If the PV is going away from the SP, and thus the error is increasing, the derivative will become stronger to pull it towards the SP.



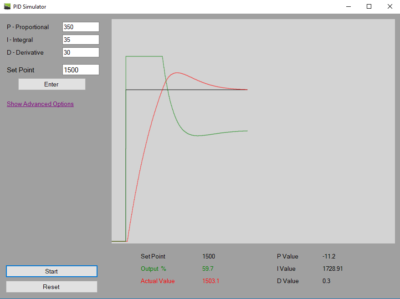
https://pidexplained.com/pid-controller-explained/

### Output

Altogether, the combination of these three relationships creates a controlled algorithm, able to reach a numerical goal accurately and precisely without overshooting.

Output = P + It + D

However, it isn’t always perfect. PID controller tuning is a fine and delicate process, where the kP, kI and kD values need to be at the right ratio to each other, and the right scale compared to your PV and SP.



https://pidexplained.com/pid-controller-explained/

## Gap Analysis

To be able to efficiently change between the different model UIs, I would need an object on the screen that can change between the different UI items and different events called on each one. This is exactly what User Controls are made for.

User Controls are a UI object that can be placed onto a canvas and dynamically changed between pre-saved User Control .xaml and .xaml.cs files. This is helpful as I can just have one button to change the current User Control, and suddenly the entire UI has changed.

The main issue with this approach is that I have absolutely no idea where to start. I only know User Controls exist after a peer of mine used them and recommended them to me to solve this very issue, so a substantial amount of research is necessary.

I need to have a User Control for Model 1, 2 and 3 and an extra for the navigation menu - that I would return to upon push of a button in each model. I’ll also need a MainWindow with just a single blank canvas, to place the User Control onto, which I can then change in code. I would have to be able to access the Canvas that the User Control is placed in, from inside the User Control, but this likely won’t be an issue thanks to being able to access an objects Parent.

## Prototype

### MainWindow.xaml

<Window x:Class="Prototype\_UI.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:Prototype\_UI"

mc:Ignorable="d"

Title="MainWindow" Height="450" Width="800">

<Grid>

<Canvas x:Name="MainCanvas"/>

</Grid>

</Window>

### MainWindow.Xaml.Cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace Prototype\_UI

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

UserControl Model;

public MainWindow()

{

InitializeComponent();

Model = new NavMenu();

Canvas.SetLeft(Model, 0);

Canvas.SetTop(Model, 0);

MainCanvas.Children.Add(Model);

}

}

}

### NavMenu.Xaml

<UserControl x:Class="Prototype\_UI.NavMenu"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:Prototype\_UI"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="160"/>

<ColumnDefinition Width="160"/>

<ColumnDefinition Width="160"/>

<ColumnDefinition Width="160"/>

<ColumnDefinition Width="160"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="160"/>

<RowDefinition Height="50"/>

<RowDefinition Height="80"/>

<RowDefinition Height="160"/>

</Grid.RowDefinitions>

<TextBlock Text="Navigation Menu" Grid.Column="2" Grid.Row="1"/>

<Button x:Name="ModelOne" Content="User Control 1" Grid.Column="1" Grid.Row="2" Click="ModelOne\_Click"/>

<Button x:Name="ModelTwo" Content="User Control 2" Grid.Column="2" Grid.Row="2" Click="ModelTwo\_Click"/>

<Button x:Name="ModelThree" Content="User Control 3" Grid.Column="3" Grid.Row="2" Click="ModelThree\_Click"/>

</Grid>

</UserControl>

### NavMenu.Xaml.Cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace Prototype\_UI

{

/// <summary>

/// Interaction logic for NavMenu.xaml

/// </summary>

public partial class NavMenu : UserControl

{

UserControl Model;

public NavMenu()

{

UserControl Model = new UserControl();

InitializeComponent();

}

private void ModelOne\_Click(object sender, RoutedEventArgs e)

{

Model = new Model1();

Canvas.SetLeft(Model, 0);

Canvas.SetTop(Model, 0);

Canvas MainCanvas = (Canvas)this.Parent;

MainCanvas.Children.Clear();

MainCanvas.Children.Add(Model);

}

private void ModelTwo\_Click(object sender, RoutedEventArgs e)

{

Model = new Model2();

Canvas.SetLeft(Model, 0);

Canvas.SetTop(Model, 0);

Canvas MainCanvas = (Canvas)this.Parent;

MainCanvas.Children.Clear();

MainCanvas.Children.Add(Model);

}

private void ModelThree\_Click(object sender, RoutedEventArgs e)

{

Model = new Model3();

Canvas.SetLeft(Model, 0);

Canvas.SetTop(Model, 0);

Canvas MainCanvas = (Canvas)this.Parent;

MainCanvas.Children.Clear();

MainCanvas.Children.Add(Model);

}

}

}

### Model1.Xaml

<UserControl x:Class="Prototype\_UI.Model1"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:Prototype\_UI"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800">

<Grid>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="160"/>

<ColumnDefinition Width="160"/>

<ColumnDefinition Width="160"/>

<ColumnDefinition Width="160"/>

<ColumnDefinition Width="160"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="160"/>

<RowDefinition Height="50"/>

<RowDefinition Height="80"/>

<RowDefinition Height="160"/>

</Grid.RowDefinitions>

<TextBlock Text="User Control 1" Grid.Column="2" Grid.Row="1"/>

<Button x:Name="GoBack" Content="Go Back" Grid.Column="2" Grid.Row="2" Click="GoBack\_Click" />

</Grid>

</Grid>

</UserControl>

### Model1.Xaml.Cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace Prototype\_UI

{

/// <summary>

/// Interaction logic for Model1.xaml

/// </summary>

public partial class Model1 : UserControl

{

public Model1()

{

InitializeComponent();

}

private void GoBack\_Click(object sender, RoutedEventArgs e)

{

UserControl Model = new NavMenu();

Canvas.SetLeft(Model, 0);

Canvas.SetTop(Model, 0);

Canvas MainCanvas = (Canvas)this.Parent;

MainCanvas.Children.Clear();

MainCanvas.Children.Add(Model);

}

}

}

### Model 2 and 3

Model 1, 2 and 3 are the same just with respective text.

### Summary

Using User Controls, I have a system where I can change between four .Xaml UI screens showing onto a blank canvas in the Main Window. This is helpful because it means I can expand upon this, putting whatever I like in the other three User Controls, as long as one is a navigation menu and the other three have a button to set the user control back to the NavMenu. The only complication that I had to solve was accessing the User Control object that the specific User Control was assigned to. To do this I could use This.Parent to access the Canvas that the User Control was placed onto, but due to it returning a generic object I had to use (Canvas) to make it believe it was a canvas and be able to access it’s children, and make changes to the user control I was currently in.

# Functional Design

## UI Design

### UI Sketches

Graphical user interface

Description automatically generated with medium confidence

The left side is a simple idea of the model across most the screen, and a column of feedback on the right side of the screen.

The right side is a bit more complex, involving opening multiple windows so that the screen with the active model is separate from the feedback, allowing you to show either one individually.

|  |  |  |
| --- | --- | --- |
|  | UI One | UI Two |
| Pros | Simple and easy to understand.  Conveys all the information needed | Allows greater control of drawing attention to certain elements, useful for a teaching tool.  Easily expandable |
| Cons | Shows everything, or nothing | A lot more complicated and could be harder for a user to understand |

So the second one does come with its advantages, like being easily expandable, but it is also going to be moderately difficult to be able to do. I have no prior experience with running multiple windows, but that hasn’t stopped me previously in this project. The other debate is whether it is worth the effort for the small number of benefits, and possible unnecessary complications.

The first model is by far simpler, extremely easy to just read off and understand, and will be a far less challenging UI to create when it comes to coding. To consider the teaching tool aspect of the program, it will need to be able to demonstrate what is going on behind the scenes clearly and accurately, so adding unnecessary complications is a significant draw back.

Overall, I think the first UI design is probably my better choice due to the simplicity and ability to clearly and easily convey the data from behind the scenes.

### Timeline Description automatically generatedFull Design

Chart

Description automatically generated

Diagram

Description automatically generated with medium confidenceChart, scatter chart

Description automatically generated

From the Main Menu (top left) you can use the three “Model x” buttons to change to Model 1 (top right), Model 2 (bottom left) or Model 3 (bottom right). On all of these the “Nav button” in the bottom right corner will lead you back to the Main Menu. On top of this, they all have a button to toggle the PID algorithm and three sliders for each of the PID constants, alongside a list of text boxes that will relay values from the current calculations.

Model 1 will use the left and right arrow keys to control the rotation of the blue rectangle. PID will control the angular acceleration of the rectangle to try keep it in an upright position, like it were modelling automatic control of a rocket.

Model 2 will have two separate displays to demonstrate 3D movement to any position in a 2D range, shown by the green circle. The blue and red arms are the same between both diagrams, simply the left is top-down and the right is side on. The user can click on any position to create a pointer that the PID algorithm will attempt to navigate the robotic arms to.

Model 3 will use PID to control rotation and velocity to create an unanchored navigation around an unlimited 2D range, like automatic driving of a car. The user can click on any position to create a pointer that the PID algorithm will attempt to navigate the red car towards.

## Hierarchy Diagram

Diagram

Description automatically generated

Generic Model contains a range of variable definitions that all other models require, for instance the kP, kI and kD constants. Due to the relationship between the ModelX.xaml and ModelX.xaml.cs, each of the models must inherit from User Control; this is easily done having GenericModel inherit from User Control.

The PID class is completely independent of any inheritance with the other structure, the only relation between them is that each model creates an instance of the class.

## Data model algorithms

Error = SetPoint - ProcessValue

P = kP \* Error

I = kI \* Error \* timeSinceLastUpdate

It = It + I

D = kD \* (Error - PreviousError) / timeSinceLastUpdate

PreviousError = Error

output = P+It+D

This algorithm is the backbone of the entire project. This should work as a simplification of Proportional, Integral and Derivative calculations.

P will only depend on the error.  
  
I will only depend upon the error, and it will continually increase until the error is negative, at which point it will start to reduce again (like the area under a curve will only increase until the curve is in the negative region)  
  
D will depend upon the error and previous error, so it can calculate if it is getting closer or further from its destination, at which point it will try slow down based on the magnitude of its velocity, independent of direction.

In the bellow example, the constants are set to 2, 1 and 3. The top row represents the state of everything going into the PID algorithm, and from there I trace through the algorithm to find the output. Ideally, there should be a large positive acceleration to counteract the current large negative velocity (it has travelled 100u in 1s)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SetPoint | ProcessValue | Error | kP | kI | kD | P | I | It | D | Previous Error | Time Since Last Update |
| 200 | 0 |  | 2 | 1 | 3 |  |  | 100 |  | 100 | 1 |
|  |  | 200 |  |  |  | 400 | 200 | 300 | 300 |  |  |

Output = P+It+D = 400+300+300=1000  
This is exactly what I was hoping for. This shows that the algorithm works perfectly under this scenario.

# Technical Design

## UML

Here is a UML representation of my project:Diagram, schematic

Description automatically generated

Diagram, schematic

Description automatically generated

## Class Breakdown

### Preamble

All the models will have a TimerEvent function that will be called x times per second – this allows me to regularly update all of the current variables and the UI. All of the TimerEvents will do almost the same thing, but they are slightly different enough for each Model that it is worth showing their pseudo code all separately.

TimerEvents will be called every (period) milliseconds, and all of them will then

* Call PID.Next
* Apply Acceleration, Velocity and Resistance
* Update the UI accordingly.

Due to each model having different UI Elements, different items controlled by PID etc. they all actually end up vastly different in size, as shown in the size of the pseudo code bellow.

### Generic Model:

#### Variables

|  |  |  |
| --- | --- | --- |
| Name | Type | Purpose |
| Timer | DispatchTimer | A variable used to define the TimerEvent() function will be called regularly. |
| DesiredD | Double | The Desired Distance |
| desiredTheta | Double | The Desired Angle |
| currentD | Double | The Current Distance |
| currentTheta | Double | The Current Angle |
| DVel | Double | The rate of change of Displacement |
| DAcc | Double | The rate of change of Velocity |
| TVel | Double | The rate of change of Angle |
| TAcc | Double | The rate of change of Angular Velocity |
| pidActive | Boolean | A true or false for whether the PID algorithm is being ran within the TImerEvent() |
| Period | Integer | The TimerEvent() will be called every x milliseconds |
| kP | Double | The Proportional Constant |
| kI | Double | The Integral Constant |
| kD | Double | The Differential Constant |
| pidTiming | Double | How many times the TimerEvent() is called per second. This is used in the PID algorithm to calculate the Integral and Derivative |

#### Functions

|  |  |
| --- | --- |
| Name | Purpose |
| PID\_Click() | Called when the PID button is clicked. Toggles PIDActive |
| PSliderValueChanged() | Called when the P Slider is changed. Sets the kP variable to the new value |
| ISliderValueChanged() | Called when the I Slider is changed. Sets the kI variable to the new value |
| DSliderValueChanged() | Called when the D Slider is changed. Sets the kD variable to the new value |
| GoBackClick() | Called when the Go Back button is clicked. Sets the MainCanvas back to the NavMenu |

### NavMenu

#### Functions

|  |  |
| --- | --- |
| Name | Purpose |
| ModelOneClick() | Changes the MainCanvas UserControl to ModelOne |
| ModelTwoClick() | Changes the MainCanvas UserControl to ModelTwo |
| ModelThreeClick() | Changes the MainCanvas UserControl to ModelThree |

### PID

#### Variables

|  |  |  |
| --- | --- | --- |
| Name | Type | Purpose |
| It | Double | Cumulative of the Integral calculations |
| PreviousError | Double | The Error variable from the previous calculation |
| Error | Double | The Difference between the Set Point and Process Value |
| ScaleValue | Double | Used in the Scale function to divide values down |
| ClampValue | Double | Used in the Clamp function to limit values |

#### Functions

|  |  |
| --- | --- |
| Name | Purpose |
| Scale | Divides by the ScaleValue |
| Clamp | If the modulus of the inputted value is greater than ClampValue, it sets the value to the corresponding positive or negative maximum. |
| Next | Runs the PID algorithm (Psuedo Code below) |

##### Next – Psuedo Code

Error = SetPoint - ProcessValue

P = kP \* Error

I = kI \* Error \* timeSinceLastUpdate

It = It + I

D = kD \* (Error - PreviousError) / timeSinceLastUpdate;

PreviousError = Error;

output = P+It+D;

Scale(output);  
Clamp(output);

### Model 1 – Rotation:

#### Variables

|  |  |  |
| --- | --- | --- |
| Name | Type | Purpose |
| PID | PID | An instance of the PID class |
| appliedTAcc | Double | Added to TAcc – The acceleration caused by the user input |
| pidTAcc | Double | Added to TAcc – The acceleration caused by the PID Algorithm |

#### Functions

|  |  |
| --- | --- |
| Name | Purpose |
| KeyDown() | Called when a key is pressed down. Checks if it is a left or right arrow key and applies the corresponding acceleration to appliedTAcc |
| KeyUp() | Called when a key is released. Checks if it is a left or right arrow key and undoes the applied affects |
| TimerEvent() | Called at regular intervals. Calls PID.Next before updating the movement variables and UI accordingly |

##### TimerEvent() Psuedo Code

if (pidActive)

{

pidRotAcc = PID.next(0, currentTheta, kP, kI, kD, pidTiming)

}

else

{

pidRotAcc = 0

}

TVel = TVel + appliedRotAcc + pidRotAcc

currentTheta = (currentTheta + TVel) % 180

Rectangle.Rotation = currentTheta

if (TVel > 0.1)

{

TVel -= 0.05

}

else if (TVel < -0.1)

{

TVel += 0.05

}

else

{

TVel = 0

}

AngleDisplay.Text = "Angle: " + Math.Round(currentTheta, 3)

RotVelDisplay.Text = "RotVel: " + Math.Round(TVel, 3)

PidAccDisplay.Text = "PidAcc: " + Math.Round(pidRotAcc, 3)

AppliedAccDisplay.Text = "AppliedAcc: " + Math.Round(appliedRotAcc, 3)

### Model 2 – Position:

#### Variables

|  |  |  |
| --- | --- | --- |
| Name | Type | Purpose |
| anglePID | PID | An instance of the PID class. Used for the angle |
| distancePID | PID | An instance of the PID class. Used for the distance |

#### Functions

|  |  |
| --- | --- |
| Name | Purpose |
| MouseLeftButtonDown() | Triggers when a position is clicked within the Range shape. Sets the desired variables in accordance to the clicked position. |
| TimerEvent() | Called at regular intervals. Calls PID.Next before updating the movement variables and UI accordingly |

##### TimerEvent() Psuedo Code

if (pidActive)

{

DAcc = distancePID.next(desiredD, currentD, kP, kI, kD, pidTiming)

if (currentTheta < desiredTheta - Math.PI)

{

TAcc = anglePID.next(desiredTheta - 2 \* Math.PI, currentTheta, kP, kI, kD, pidTiming)

}

else if (currentTheta - Math.PI > desiredTheta)

{

TAcc = anglePID.next(desiredTheta, currentTheta - 2 \* Math.PI, kP, kI, kD, pidTiming)

}

else

{

TAcc = anglePID.next(desiredTheta, currentTheta, kP, kI, kD, pidTiming)

}

}

else

{

DAcc = 0

TAcc = 0

}

TVel = Math.Min(TVel + TAcc, Math.PI / 10)

if (DVel + DAcc > 0)

{

DVel = Math.Min(DVel + DAcc, 2)

}

else

{

DVel = Math.Max(DVel + DAcc, -2)

}

currentTheta = (currentTheta + TVel)

while (currentTheta < 0)

{

currentTheta += 2 \* Math.PI

}

currentD = Math.Min(Math.Abs(currentD + DVel), 200)

if (Math.Abs(currentD) == 200)

{

DVel = 0

}

if (TVel > 0)

{

TVel = 0.98 \* TVel

}

else if (TVel < 0)

{

TVel = 0.98 \* TVel

}

else if (TVel < 0.01 && TVel > -0.01)

{

TVel = 0

}

if (TVel > 0.25)

{

TVel = 0.25

}

else if (TVel < -0.25)

{

TVel = -0.25

}

if (DVel > 0.1)

{

DVel = 0.98 \* DVel

}

else if (DVel < -0.1)

{

DVel = 0.98 \* DVel

}

else

{

DVel = 0

}

DesiredThetaDisplay.Text = "Desired Angle: " + Convert.ToString(Math.Round(desiredTheta, 2))

DesiredDDisplay.Text = "Desired Distance: " + Convert.ToString(Math.Round(desiredD, 2))

CurrentThetaDisplay.Text = "Current Angle: " + Convert.ToString(Math.Round(currentTheta, 2))

CurrentDDisplay.Text = "Current Distance: " + Convert.ToString(Math.Round(currentD, 2))

ThetaVelDisplay.Text = "Theta Velocity: " + Convert.ToString(Math.Round(TVel, 3))

ThetaAccelerationDisplay.Text = "Theta Acceleration: " + Convert.ToString(Math.Round(TAcc, 4))

DistanceVelocityDisplay.Text = "Distance Velocity: " + Convert.ToString(Math.Round(DVel, 3))

DistanceAccelerationDisplay.Text = "Distance Acceleration: " + Convert.ToString(Math.Round(DAcc, 4))

topDownArmOne.X2 = topDownArmOne.X1 + (currentD / 2) \* Math.Cos(currentTheta)

topDownArmOne.Y2 = topDownArmOne.Y1 + (currentD / 2) \* Math.Sin(currentTheta)

topDownArmTwo.X1 = topDownArmOne.X2

topDownArmTwo.Y1 = topDownArmOne.Y2

topDownArmTwo.X2 = topDownArmOne.X1 + currentD \* Math.Cos(currentTheta)

topDownArmTwo.Y2 = topDownArmOne.Y1 + currentD \* Math.Sin(currentTheta)

sideOnArmOne.X2 = sideOnArmOne.X1 + currentD / 2

sideOnArmOne.Y2 = sideOnArmOne.Y1 - Math.Sqrt((100 \* 100) - (currentD \* currentD / 4))

sideOnArmTwo.X1 = sideOnArmOne.X2

sideOnArmTwo.Y1 = sideOnArmOne.Y2

sideOnArmTwo.X2 = sideOnArmOne.X1 + currentD

### Model 3 – Velocity:

#### Variables

|  |  |  |
| --- | --- | --- |
| Name | Type | Purpose |
| anglePID | PID | An instance of the PID class. Used for the angle |
| distancePID | PID | An instance of the PID class. Used for the distance |

#### Functions

|  |  |
| --- | --- |
| Name | Purpose |
| MouseLeftButtonDown() | Triggers when any position is clicked. Sets the desired variables in accordance to the clicked position. |
| TimerEvent() | Called at regular intervals. Calls PID.Next before updating the movement variables and UI accordingly |

##### TimerEvent()

deltaX = desiredX - currentX

deltaY = desiredY - currentY

desiredD = Math.Sqrt(deltaX \* deltaX + deltaY \* deltaY)

if (deltaY >= 0)

{

desiredTheta = Math.PI - Math.Atan(deltaX / deltaY)

}

else

{

if (deltaX >= 0)

{

desiredTheta = -Math.Atan(deltaX / deltaY)

}

else

{

desiredTheta = 2 \* Math.PI - Math.Atan(deltaX / deltaY)

}

}

if (pidActive)

{

DAcc = distancePID.next(desiredD, 0, kP, kI, kD, pidTiming)

if (desiredD < 1)

{

currentX = desiredX

currentY = desiredY

DAcc = 0

DVel = 0

desiredTheta = 0.5\*Math.PI

}

if (0 <= currentTheta && currentTheta <= 0.5 \* Math.PI && 1.5 \* Math.PI <= desiredTheta && desiredTheta <= 2 \* Math.PI)

{

TAcc = anglePID.next(desiredTheta, currentTheta + 2 \* Math.PI, kP, kI, kD, pidTiming)

}

else if (0 <= desiredTheta && desiredTheta <= 0.5 \* Math.PI && 1.5 \* Math.PI <= currentTheta && currentTheta <= 2 \* Math.PI)

{

TAcc = anglePID.next(desiredTheta + 2 \* Math.PI, currentTheta, kP, kI, kD, pidTiming)

}

else

{

TAcc = anglePID.next(desiredTheta, currentTheta, kP, kI, kD, pidTiming)

}

}

else

{

DAcc = 0

TAcc= 0

}

if (TVel >= 0)

{

TVel = Math.Min(TVel + TAcc, Math.PI / 20)

}

else

{

TVel = Math.Max(TVel + TAcc, -Math.PI / 10)

}

DVel = Math.Min(DVel + DAcc, 2)

currentTheta = (currentTheta + TVel)%(2\*Math.PI)

while(currentTheta < 0)

{

currentTheta += 2\*Math.PI

}

if (currentTheta >= 1.5\*Math.PI)

{

currentX -= DVel\*Math.Cos(currentTheta-1.5\*Math.PI)

currentY -= DVel\*Math.Sin(currentTheta-1.5\*Math.PI)

}

else if(currentTheta >= Math.PI)

{

currentX -= DVel \* Math.Sin(currentTheta - Math.PI)

currentY += DVel \* Math.Cos(currentTheta - Math.PI)

}

else if (currentTheta >= 0.5 \* Math.PI)

{

currentX += DVel \* Math.Cos(currentTheta - 0.5 \* Math.PI)

currentY += DVel \* Math.Sin(currentTheta - 0.5 \* Math.PI)

}

else

{

currentX += DVel \* Math.Cos(currentTheta)

currentY -= DVel \* Math.Sin(currentTheta)

}

TVel \*= 0.98

if (TVel > 0.25)

{

TVel = 0.25

}

else if (TVel < -0.25)

{

TVel = -0.25

}

else if(-0.001 < TVel && TVel < 0.001)

{

TVel = 0

}

if (DVel > 0.01)

{

DVel = 0.98 \* DVel

}

else

{

DVel = 0

}

Canvas.SetLeft(Car, currentX-50)

Canvas.SetTop(Car, currentY-31.25)

Car.Rotation = 360 \* currentTheta / (2 \* Math.PI) - 90

DesiredDDisplay.Text = "Desired Distance: " + Math.Round(desiredD,2)

DesiredThetaDisplay.Text = "Desired Angle: " + Math.Round(desiredTheta/Math.PI,2) + " Pi"

CurrentThetaDisplay.Text = "Current Angle: " + Math.Round(currentTheta/Math.PI,2) +" Pi"

ThetaVelDisplay.Text = "Theta Velocity: " + Math.Round(TVel,7)

ThetaAccelerationDisplay.Text = "Theta Acceleration: " + Math.Round(TAcc,7)

DistanceVelocityDisplay.Text = "Distance Velocity: " + Math.Round(DVel,2)

DistanceAccelerationDisplay.Text = "Distance Acceleration: " + Math.Round(DAcc,2)

# Technical Solution

An easier view of the code is available at  
<https://github.com/Lukewal99/Y13Project>  
All annotations are notes in the code – so all annotations are still visible there.

## PID.PID.cs

using System;

namespace PID

{

public class PID

{

// Define Variables

public double It = 0;

private double PreviousError = 0;

private double Error = 0;

double ScaleValue = 0;

double ClampValue = 0;

public PID(double ScaleValueIn, double ClampValueIn)

{

// Set upon creation

ScaleValue = ScaleValueIn;

ClampValue = ClampValueIn;

}

public double next(double SetPoint, double ProcessValue, double kP, double kI, double kD, double timeSinceLastUpdate)

{ //Desired Value, Current Value, constant P, I, D, time since last update

// Difference between Desired and Current

Error = SetPoint - ProcessValue;

// P

double P = kP \* Error;

// I

double I = kI \* Error \* timeSinceLastUpdate;

this.It += I;

// D

double D = kD \* (Error - PreviousError) / timeSinceLastUpdate;

//Scale and Clamp D

D = Scale(D, ScaleValue);

D = Clamp(D, ClampValue/3);

this.PreviousError = Error;

// calculate output

double output = P+It+D;

// Scale and Clamp Output

output = Scale(output, ScaleValue);

output = Clamp(output, ClampValue);

return output;

}

private double Scale(double input, double ScaleValue)

{

// Scale

double output = input / ScaleValue;

return output;

}

private double Clamp(double input, double ClampValue)

{

// Clamp

if (input > ClampValue)

{

return ClampValue;

}

else if (input < -ClampValue)

{

return -ClampValue;

}

else if (0.001/ScaleValue < input && input < 0.001/ScaleValue)

{

// Approaching Zero

return 0;

}

else

{

return input;

}

}

}

}

## GenericModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Controls;

using System.Windows.Threading;

using NavMenuNew;

namespace NavMenuNew

{

public class GenericModel : UserControl

{

// Define Variables

public DispatcherTimer timer;

public double desiredD = 0;

public double desiredTheta = Math.PI / 2;

public double currentD = 0;

public double currentTheta = 0;

public double DVel = 0;

public double DAcc = 0;

public double TVel = 0;

public double TAcc = 0;

public bool pidActive = false;

public int Period = 10; //The loop will run every x milliseconds

public double kP = 0;

public double kI = 0;

public double kD = 0;

public double pidTiming = 0;

}

}

## MainWindow.xaml

<Window x:Class="NavMenuNew.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:NavMenuNew"

mc:Ignorable="d"

Title="MainWindow" Height="500" Width="850">

<Canvas x:Name="MainCanvas"/>

</Window>

## MainWindow.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace NavMenuNew

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

UserControl Model;

public MainWindow()

{

// Create the Nav Menu and set the canvas to it

InitializeComponent();

Model = new NavMenu();

Canvas.SetLeft(Model, 0);

Canvas.SetTop(Model, 0);

MainCanvas.Children.Add(Model);

}

}

}

## NavMenu.xaml

<UserControl x:Class="NavMenuNew.NavMenu"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:NavMenuNew"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800">

<Grid>

<Grid Height="450" Width="850" HorizontalAlignment="Center" VerticalAlignment="Top">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="100"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="100"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="\*"/>

<RowDefinition Height="50"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<TextBlock Text="PID" FontSize="50" Foreground="DarkOrange" FontFamily="Times New Roman" FontWeight="Black" Grid.Column="2" HorizontalAlignment="Center" VerticalAlignment="Center"/>

<Button x:Name="ModelOne" Content="MODEL 1: ROTATION" Grid.Column="1" Grid.Row="1" Click="ModelOne\_Click" />

<Button x:Name="ModelTwo" Content="MODEL 2: POSITION" Grid.Column="2" Grid.Row="1" Click="ModelTwo\_Click"/>

<Button x:Name="ModelThree" Content="MODEL 3: VELOCITY" Grid.Column="3" Grid.Row="1" Click="ModelThree\_Click"/>

<Canvas>

<TextBlock Text="PID is a closed feedback loop using Proportional, Integral, and Derivative calculation, designed to meet a numerical goal without overshooting." Canvas.Top="310" Height="25" Width="840" HorizontalAlignment="Center" Canvas.Left="9" VerticalAlignment="Top" TextAlignment="Center"/>

<TextBlock Text="P = Err \* kP / I = kI \* Err \* Δt / D = kd \* (Err – pErr) ÷ Δt" Canvas.Top="335" Height="25" Width="350" HorizontalAlignment="Center" Canvas.Left="9" VerticalAlignment="Top" TextAlignment="Center"/>

<TextBlock Text="It = It + I" Canvas.Top="350" Height="25" Width="350" HorizontalAlignment="Center" Canvas.Left="9" VerticalAlignment="Top" TextAlignment="Center"/>

<TextBlock Text="Output = P + It + D" Canvas.Top="375" Height="25" Width="350" HorizontalAlignment="Center" Canvas.Left="9" VerticalAlignment="Top" TextAlignment="Center"/>

<TextBlock Text="However, it isn’t always perfect. PID controller tuning is a fine and delicate process, where the kP, kI and" Canvas.Top="400" Height="25" HorizontalAlignment="Center" Canvas.Left="9" VerticalAlignment="Top" TextAlignment="Center" Width="845"/>

<TextBlock Text="kD values need to be at the right ratio to each other, and the right scale compared to your PV and SP." Canvas.Top="415" Height="25" HorizontalAlignment="Center" Canvas.Left="9" VerticalAlignment="Top" TextAlignment="Center" Width="845"/>

<TextBlock Text="Proportional will accelerate towards the Set Point proportionally to the Error." Canvas.Top="335" Height="55" Width="475" Canvas.Left="364" TextAlignment="Center" HorizontalAlignment="Center" VerticalAlignment="Top"/>

<TextBlock Text="Integral will increase acceleration the longer it takes to reach the Set Point." Canvas.Top="350" Height="55" Width="475" Canvas.Left="364" TextAlignment="Center" HorizontalAlignment="Center" VerticalAlignment="Top"/>

<TextBlock Text="Derrivative will try and deccelerate before it crosses the Set Point, and stop on point." Canvas.Top="365" Height="55" Width="475" Canvas.Left="364" TextAlignment="Center" HorizontalAlignment="Center" VerticalAlignment="Top"/>

</Canvas>

</Grid>

</Grid>

</UserControl>

## NavMenu.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace NavMenuNew

{

/// <summary>

/// Interaction logic for NavMenu.xaml

/// </summary>

public partial class NavMenu : UserControl

{

UserControl Model;

public NavMenu()

{

UserControl Model = new UserControl();

InitializeComponent();

}

private void ModelOne\_Click(object sender, RoutedEventArgs e)

{

// Change to Model One

Model = new Model1();

Canvas.SetLeft(Model, 0);

Canvas.SetTop(Model, 0);

Canvas MainCanvas = (Canvas)this.Parent;

MainCanvas.Children.Clear();

MainCanvas.Children.Add(Model);

}

private void ModelTwo\_Click(object sender, RoutedEventArgs e)

{

// Change to Model Two

Model = new Model2();

Canvas.SetLeft(Model, 0);

Canvas.SetTop(Model, 0);

Canvas MainCanvas = (Canvas)this.Parent;

MainCanvas.Children.Clear();

MainCanvas.Children.Add(Model);

}

private void ModelThree\_Click(object sender, RoutedEventArgs e)

{

// Change to Model Three

Model = new Model3();

Canvas.SetLeft(Model, 0);

Canvas.SetTop(Model, 0);

Canvas MainCanvas = (Canvas)this.Parent;

MainCanvas.Children.Clear();

MainCanvas.Children.Add(Model);

}

}

}

## RotationModel.xaml

<local:GenericModel x:Class="NavMenuNew.Model1"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:NavMenuNew"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800">

<Grid KeyDown="Grid\_KeyDown" KeyUp="Grid\_KeyUp">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="700"/>

<ColumnDefinition Width="100"/>

</Grid.ColumnDefinitions>

<Canvas Width="800" Height="450" Grid.Column="0" Background="White">

<TextBlock x:Name="AngleDisplay" Text="Angle: 0" Canvas.Left="0" Canvas.Top="10"/>

<TextBlock x:Name="RotVelDisplay" Text="RotVel: 0" Canvas.Left="0" Canvas.Top="30"/>

<TextBlock x:Name="PidAccDisplay" Text="PidAcc: 0" Canvas.Left="0" Canvas.Top="50"/>

<TextBlock x:Name="AppliedAccDisplay" Text="AppliedAcc: 0" Canvas.Left="0" Canvas.Top="70"/>

<TextBlock Text="Double Press PID ACTIVE for keys to work" Canvas.Left="275" Canvas.Top="377" Width="250" Height="20" TextAlignment="Center" HorizontalAlignment="Center" VerticalAlignment="Top"/>

<TextBlock Text="Left or right arrows to turn" Canvas.Left="325" Canvas.Top="390" Width="150" Height="20" TextAlignment="Center"/>

<TextBlock Text="0 to reset" Canvas.Left="325" Canvas.Top="405" Width="150" Height="20" TextAlignment="Center"/>

<Rectangle x:Name="Rectangle" Fill="Blue" Width="80" Height="200" Canvas.Left="360" Canvas.Top="125" RenderTransformOrigin="0.5,0.5">

<Rectangle.RenderTransform>

<RotateTransform Angle="0"/>

</Rectangle.RenderTransform>

</Rectangle>

</Canvas>

<Grid Grid.Column="1" Background="DarkGray">

<Grid.RowDefinitions>

<RowDefinition Height="50"/>

<RowDefinition Height="20"/>

<RowDefinition Height="330"/>

<RowDefinition Height="50"/>

</Grid.RowDefinitions>

<Button x:Name="PidActive" Content="PID Active?" Grid.Row="0" Click="PidActive\_Click"/>

<TextBlock x:Name="PidActiveDisplay" Text="False" Grid.Row="1" HorizontalAlignment="Center" Foreground="Red"/>

<Button x:Name="GoBack" Content="Go Back" Grid.Row="3" Click="GoBack\_Click" />

<Grid Grid.Row="2">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="20"/>

<RowDefinition Height="\*"/>

<RowDefinition Height="20"/>

<RowDefinition Height="20"/>

</Grid.RowDefinitions>

<TextBlock Text="P" HorizontalAlignment="Center" Grid.Row="0" Grid.Column="0"/>

<TextBlock Text="I" HorizontalAlignment="Center" Grid.Row="0" Grid.Column="1"/>

<TextBlock Text="D" HorizontalAlignment="Center" Grid.Row="0" Grid.Column="2"/>

<Slider x:Name="PSlider" Grid.Row="1" Grid.Column="0" Orientation="Vertical" HorizontalAlignment="Center" Maximum="10" Minimum="0" ValueChanged="PSlider\_ValueChanged" TickFrequency="0.1" SmallChange="0.1" IsSnapToTickEnabled="True"/>

<Slider x:Name="ISlider" Grid.Row="1" Grid.Column="1" Orientation="Vertical" HorizontalAlignment="Center" Maximum="10" Minimum="0" ValueChanged="ISlider\_ValueChanged" TickFrequency="0.1" SmallChange="0.1" IsSnapToTickEnabled="True"/>

<Slider x:Name="DSlider" Grid.Row="1" Grid.Column="2" Orientation="Vertical" HorizontalAlignment="Center" Maximum="10" Minimum="0" ValueChanged="DSlider\_ValueChanged" TickFrequency="0.1" SmallChange="0.1" IsSnapToTickEnabled="True"/>

<TextBlock x:Name="PValue" Grid.Row="2" Grid.Column="0" Text="0" HorizontalAlignment="Center"/>

<TextBlock x:Name="IValue" Grid.Row="2" Grid.Column="1" Text="0" HorizontalAlignment="Center"/>

<TextBlock x:Name="DValue" Grid.Row="2" Grid.Column="2" Text="0" HorizontalAlignment="Center"/>

</Grid>

</Grid>

</Grid>

</local:GenericModel>

## RotationModel.xaml.cs

using NavMenuNew;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Windows.Threading;

namespace NavMenuNew

{

/// <summary>

/// Interaction logic for Model1.xaml

/// </summary>

public partial class Model1 : GenericModel

{

// Define variables

public double appliedRotAcc = 0;

public double pidRotAcc = 0;

PID.PID PID = new PID.PID(250, 0.5);

public Model1()

{

InitializeComponent();

// Set up timer event

timer = new DispatcherTimer();

pidTiming = Period / 1000F;

timer.Interval = new TimeSpan(0, 0, 0, 0, Period);

timer.Tick += TimerEvent;

timer.Start();

}

private void TimerEvent(object sender, EventArgs e)

{

// Calculate PID

// set TAcc to 0 if !pidActive

if (pidActive)

{

pidRotAcc = PID.next(0, currentTheta, kP, kI, kD, pidTiming);

}

else

{

pidRotAcc = 0;

}

// Update Angle

TVel = TVel + appliedRotAcc + pidRotAcc;

currentTheta = (currentTheta + TVel) % 180;

// Update rotation on the UI

RotateTransform rotateTransform = new RotateTransform(currentTheta);

Rectangle.RenderTransform = rotateTransform;

// Apply resistance

if (TVel > 0.1)

{

TVel -= 0.05;

}

else if (TVel < -0.1)

{

TVel += 0.05;

}

else

{

TVel = 0;

}

AngleDisplay.Text = "Angle: " + Math.Round(currentTheta, 3);

RotVelDisplay.Text = "RotVel: " + Math.Round(TVel, 3);

PidAccDisplay.Text = "PidAcc: " + Math.Round(pidRotAcc, 3);

AppliedAccDisplay.Text = "AppliedAcc: " + Math.Round(appliedRotAcc, 3);

}

private void PidActive\_Click(object sender, RoutedEventArgs e)

{

// Flip pidActive

pidActive = !pidActive;

PidActiveDisplay.Text = Convert.ToString(pidActive);

if (pidActive) // Update UI

{

PidActiveDisplay.Foreground = new SolidColorBrush(Color.FromRgb(0, 185, 0));

PID.It = 0;

}

else if (!pidActive)

{

PidActiveDisplay.Foreground = new SolidColorBrush(Color.FromRgb(255, 0, 0));

}

}

private void PSlider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

// P Changed

kP = Math.Round(Convert.ToDouble(PSlider.Value), 2);

PValue.Text = Convert.ToString(kP);

}

private void ISlider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

// I Changed

kI = Math.Round(Convert.ToDouble(ISlider.Value), 3);

IValue.Text = Convert.ToString(kI);

}

private void DSlider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

// D Changed

kD = Math.Round(Convert.ToDouble(DSlider.Value), 4);

DValue.Text = Convert.ToString(kD);

}

private void GoBack\_Click(object sender, RoutedEventArgs e)

{

// Go back to NavMenu

UserControl Model = new NavMenu();

Canvas.SetLeft(Model, 0);

Canvas.SetTop(Model, 0);

Canvas MainCanvas = (Canvas)this.Parent;

MainCanvas.Children.Clear();

MainCanvas.Children.Add(Model);

}

private void Grid\_KeyDown(object sender, KeyEventArgs e)

{

// detect key down

// check what key and apply the required effect

if (e.Key == Key.Left)

{

appliedRotAcc = -0.2;

}

else if (e.Key == Key.Right)

{

appliedRotAcc = 0.2;

}

else if (e.Key == Key.D0)

{

currentTheta = 0;

TVel = 0;

pidRotAcc = 0;

PID.It = 0;

pidActive = false;

PidActiveDisplay.Text = "False";

PidActiveDisplay.Foreground = new SolidColorBrush(Color.FromRgb(255, 0, 0));

}

}

private void Grid\_KeyUp(object sender, KeyEventArgs e)

{

// undo effects of keyDown

if (e.Key == Key.Left || e.Key == Key.Right)

{

appliedRotAcc = 0;

}

}

}

}

## PositionModel.xaml

<local:GenericModel x:Class="NavMenuNew.Model2"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:NavMenuNew"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="700"/>

<ColumnDefinition Width="100"/>

</Grid.ColumnDefinitions>

<Canvas Grid.Column="0" Grid.Row="0" x:Name="largeCanvas" Background="White">

<TextBlock Text="Desired Distance: " Canvas.Top="10" Canvas.Right="10" x:Name="DesiredDDisplay"/>

<TextBlock Text="Desired Angle: " Canvas.Top="30" Canvas.Right="10" x:Name="DesiredThetaDisplay"/>

<TextBlock Text="Current Distance: " Canvas.Top="50" Canvas.Right="10" x:Name="CurrentDDisplay"/>

<TextBlock Text="Current Angle: " Canvas.Top="70" Canvas.Right="10" x:Name="CurrentThetaDisplay"/>

<TextBlock Text="Theta Velocity: " Canvas.Top="100" Canvas.Right="10" x:Name="ThetaVelDisplay"/>

<TextBlock Text="Theta Acceleration: " Canvas.Top="120" Canvas.Right="10" x:Name="ThetaAccelerationDisplay"/>

<TextBlock Text="Distance Velocity: " Canvas.Top="140" Canvas.Right="10" x:Name="DistanceVelocityDisplay"/>

<TextBlock Text="Distance Acceleration: " Canvas.Top="160" Canvas.Right="10" x:Name="DistanceAccelerationDisplay"/>

<Ellipse Canvas.Left="425" Canvas.Top="212.5" Width="25" Height="25" Fill="Brown"/>

<Line X1="437.5" Y1="225" X2="487.5" Y2="138" Stroke="DarkBlue" x:Name="sideOnArmOne" StrokeThickness="3"/>

<Line X1="487.5" Y1="138" X2="537.5" Y2="225" Stroke="Red" x:Name="sideOnArmTwo" StrokeThickness="3"/>

<Ellipse Canvas.Left="0" Canvas.Top="25" Width="400" Height="400" Stroke="Green" Fill="White" x:Name="Range" MouseLeftButtonDown="Range\_MouseLeftButtonDown"/>

<Rectangle x:Name="pointer" Canvas.Left="195" Canvas.Top="220" Fill="ForestGreen" Height="10" Width="10" RenderTransformOrigin="0.5,0.5">

<Rectangle.RenderTransform>

<RotateTransform Angle="45"/>

</Rectangle.RenderTransform>

</Rectangle>

<Ellipse Canvas.Left="175" Canvas.Top="200" Width="50" Height="50" Fill="Brown" x:Name="topDownBase"/>

<Line X1="200" Y1="225" X2="250" Y2="225" Stroke="DarkBlue" x:Name="topDownArmOne" StrokeThickness="3"/>

<Line X1="250" Y1="225" X2="300" Y2="225" Stroke="Red" x:Name="topDownArmTwo" StrokeThickness="3"/>

</Canvas>

<Grid Grid.Column="1" Background="DarkGray">

<Grid.RowDefinitions>

<RowDefinition Height="50"/>

<RowDefinition Height="20"/>

<RowDefinition Height="330"/>

<RowDefinition Height="50"/>

</Grid.RowDefinitions>

<Button x:Name="PidActive" Content="PID Active?" Grid.Row="0" Click="PidActive\_Click"/>

<TextBlock x:Name="PidActiveDisplay" Text="False" Grid.Row="1" HorizontalAlignment="Center" Foreground="Red"/>

<Button x:Name="GoBack" Content="Go Back" Grid.Row="3" Click="GoBack\_Click" />

<Grid Grid.Row="2">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="20"/>

<RowDefinition Height="\*"/>

<RowDefinition Height="20"/>

<RowDefinition Height="20"/>

</Grid.RowDefinitions>

<TextBlock Text="P" HorizontalAlignment="Center" Grid.Row="0" Grid.Column="0"/>

<TextBlock Text="I" HorizontalAlignment="Center" Grid.Row="0" Grid.Column="1"/>

<TextBlock Text="D" HorizontalAlignment="Center" Grid.Row="0" Grid.Column="2"/>

<Slider x:Name="PSlider" Grid.Row="1" Grid.Column="0" Orientation="Vertical" HorizontalAlignment="Center" Maximum="10" Minimum="0" ValueChanged="PSlider\_ValueChanged" TickFrequency="0.1" SmallChange="0.1" IsSnapToTickEnabled="True"/>

<Slider x:Name="ISlider" Grid.Row="1" Grid.Column="1" Orientation="Vertical" HorizontalAlignment="Center" Maximum="10" Minimum="0" ValueChanged="ISlider\_ValueChanged" TickFrequency="0.1" SmallChange="0.1" IsSnapToTickEnabled="True"/>

<Slider x:Name="DSlider" Grid.Row="1" Grid.Column="2" Orientation="Vertical" HorizontalAlignment="Center" Maximum="10" Minimum="0" ValueChanged="DSlider\_ValueChanged" TickFrequency="0.1" SmallChange="0.1" IsSnapToTickEnabled="True"/>

<TextBlock x:Name="PValue" Grid.Row="2" Grid.Column="0" Text="0" HorizontalAlignment="Center"/>

<TextBlock x:Name="IValue" Grid.Row="2" Grid.Column="1" Text="0" HorizontalAlignment="Center"/>

<TextBlock x:Name="DValue" Grid.Row="2" Grid.Column="2" Text="0" HorizontalAlignment="Center"/>

</Grid>

</Grid>

</Grid>

</local:GenericModel>

## PositionModel.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Windows.Threading;

namespace NavMenuNew

{

/// <summary>

/// Interaction logic for Model2.xaml

/// </summary>

public partial class Model2 : GenericModel

{

// Define variables

PID.PID anglePID = new PID.PID(1000, 0.1);

PID.PID distancePID = new PID.PID(400, 0.5);

public Model2()

{

InitializeComponent();

// Create the timer event

timer = new DispatcherTimer();

pidTiming = Period / 1000F;

timer.Interval = new TimeSpan(0, 0, 0, 0, Period);

timer.Tick += TimerEvent;

timer.Start();

}

private void TimerEvent(object sender, EventArgs e)

{

// Calculate PID

// set DAcc and TAcc to 0 if !pidActive

if (pidActive)

{

//DAcc

DAcc = distancePID.next(desiredD, currentD, kP, kI, kD, pidTiming);

//TAcc

if (currentTheta < desiredTheta - Math.PI)

{

TAcc = anglePID.next(desiredTheta - 2 \* Math.PI, currentTheta, kP, kI, kD, pidTiming);

}

else if (currentTheta - Math.PI > desiredTheta)

{

TAcc = anglePID.next(desiredTheta, currentTheta - 2 \* Math.PI, kP, kI, kD, pidTiming);

}

else

{

TAcc = anglePID.next(desiredTheta, currentTheta, kP, kI, kD, pidTiming);

}

}

else

{

DAcc = 0;

TAcc = 0;

}

// Apply TAcc

// Cap at 1/20 pi

TVel = Math.Min(TVel + TAcc, Math.PI / 10);

// Apply DAcc

// always positive

// Cap at 2

if (DVel + DAcc > 0)

{

DVel = Math.Min(DVel + DAcc, 2);

}

else

{

DVel = Math.Max(DVel + DAcc, -2);

}

// Apply TVel

currentTheta = (currentTheta + TVel);

while (currentTheta < 0)

{

currentTheta += 2 \* Math.PI;

}

// Apply DVel

currentD = Math.Min(Math.Abs(currentD + DVel), 200);

if (Math.Abs(currentD) == 200)

{

DVel = 0;

}

// Apply Resistance

if (TVel > 0)

{

TVel = 0.98 \* TVel;

}

else if (TVel < 0)

{

TVel = 0.98 \* TVel;

}

else if (TVel < 0.01 && TVel > -0.01)

{

TVel = 0;

}

if (TVel > 0.25)

{

TVel = 0.25;

}

else if (TVel < -0.25)

{

TVel = -0.25;

}

if (DVel > 0.1)

{

DVel = 0.98 \* DVel;

}

else if (DVel < -0.1)

{

DVel = 0.98 \* DVel;

}

else

{

DVel = 0;

}

//Update UI

DesiredThetaDisplay.Text = "Desired Angle: " + Convert.ToString(Math.Round(desiredTheta, 2));

DesiredDDisplay.Text = "Desired Distance: " + Convert.ToString(Math.Round(desiredD, 2));

CurrentThetaDisplay.Text = "Current Angle: " + Convert.ToString(Math.Round(currentTheta, 2));

CurrentDDisplay.Text = "Current Distance: " + Convert.ToString(Math.Round(currentD, 2));

ThetaVelDisplay.Text = "Theta Velocity: " + Convert.ToString(Math.Round(TVel, 3));

ThetaAccelerationDisplay.Text = "Theta Acceleration: " + Convert.ToString(Math.Round(TAcc, 4));

DistanceVelocityDisplay.Text = "Distance Velocity: " + Convert.ToString(Math.Round(DVel, 3));

DistanceAccelerationDisplay.Text = "Distance Acceleration: " + Convert.ToString(Math.Round(DAcc, 4));

//Sets a pointer position to the current desired location

Canvas.SetLeft(pointer, Canvas.GetLeft(topDownBase) + topDownBase.Width / 2 - pointer.Width / 2 + desiredD \* Math.Cos(desiredTheta));

Canvas.SetTop(pointer, Canvas.GetTop(topDownBase) + topDownBase.Height / 2 - pointer.Height / 2 + desiredD \* Math.Sin(desiredTheta));

//changes arm to reflect currentTheta

topDownArmOne.X2 = topDownArmOne.X1 + (currentD / 2) \* Math.Cos(currentTheta);

topDownArmOne.Y2 = topDownArmOne.Y1 + (currentD / 2) \* Math.Sin(currentTheta);

topDownArmTwo.X1 = topDownArmOne.X2;

topDownArmTwo.Y1 = topDownArmOne.Y2;

topDownArmTwo.X2 = topDownArmOne.X1 + currentD \* Math.Cos(currentTheta);

topDownArmTwo.Y2 = topDownArmOne.Y1 + currentD \* Math.Sin(currentTheta);

//changes arms to reflect currentD

sideOnArmOne.X2 = sideOnArmOne.X1 + currentD / 2;

sideOnArmOne.Y2 = sideOnArmOne.Y1 - Math.Sqrt((100 \* 100) - (currentD \* currentD / 4));

sideOnArmTwo.X1 = sideOnArmOne.X2;

sideOnArmTwo.Y1 = sideOnArmOne.Y2;

sideOnArmTwo.X2 = sideOnArmOne.X1 + currentD;

//sideOnArmTwo.Y2 Doesnt change

}

private void PSlider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

// P Changed

kP = Math.Round(Convert.ToDouble(PSlider.Value), 2);

PValue.Text = Convert.ToString(kP);

}

private void ISlider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

// I Changed

kI = Math.Round(Convert.ToDouble(ISlider.Value), 2);

IValue.Text = Convert.ToString(kI);

}

private void DSlider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

// D Changed

kD = Math.Round(Convert.ToDouble(DSlider.Value), 2);

DValue.Text = Convert.ToString(kD);

}

private void PidActive\_Click(object sender, RoutedEventArgs e)

{

// Flip pidActive

pidActive = !pidActive;

PidActiveDisplay.Text = Convert.ToString(pidActive);

if (pidActive) // Update UI

{

PidActiveDisplay.Foreground = new SolidColorBrush(Color.FromRgb(0, 185, 0));

anglePID.It = 0;

distancePID.It = 0;

}

else if (!pidActive)

{

PidActiveDisplay.Foreground = new SolidColorBrush(Color.FromRgb(255, 0, 0));

}

}

private void GoBack\_Click(object sender, RoutedEventArgs e)

{

// Go back to NavMenu

UserControl Model = new NavMenu();

Canvas.SetLeft(Model, 0);

Canvas.SetTop(Model, 0);

Canvas MainCanvas = (Canvas)this.Parent;

MainCanvas.Children.Clear();

MainCanvas.Children.Add(Model);

}

private void Range\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

//Detect click in range

Point mousePos = e.GetPosition(Range);

double x = mousePos.X - 200;

double y = mousePos.Y - 200;

// Set desired D

desiredD = Math.Min(Math.Sqrt(x \* x + y \* y), 200);

// set desired Theta

if (x >= 0)

{

if (y >= 0)

{

desiredTheta = Math.Atan(y / x);

}

else if (y < 0)

{

desiredTheta = Math.Atan(y / x) + 2 \* Math.PI;

}

}

else if (x < 0)

{

desiredTheta = Math.Atan(y / x) + Math.PI;

}

DesiredDDisplay.Text = "Desired Distance: " + Convert.ToString(Math.Round(desiredD, 2));

DesiredThetaDisplay.Text = "Desired Angle: " + Convert.ToString(Math.Round(desiredTheta, 2));

}

}

}

## VelocityModel.xaml

<local:GenericModel x:Class="NavMenuNew.Model3"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:NavMenuNew"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="700"/>

<ColumnDefinition Width="100"/>

</Grid.ColumnDefinitions>

<Canvas Grid.Column="0" Grid.Row="0" x:Name="largeCanvas" Background="White" MouseLeftButtonDown="largeCanvas\_MouseLeftButtonDown">

<Rectangle x:Name="Car" Width="100" Height="62.5" Canvas.Left="50" Canvas.Top="69" RadiusX="0" RadiusY="0" HorizontalAlignment="Left" VerticalAlignment="Top" RenderTransformOrigin="0.5,0.5">

<Rectangle.RenderTransform>

<RotateTransform Angle="0"/>

</Rectangle.RenderTransform>

<Rectangle.Fill>

<!--<SolidColorBrush Color="red"/>-->

<ImageBrush ImageSource="C:\Users\lukew\OneDrive\Documents\GitHub\Project\NavMenuNew\Car.png"/>

</Rectangle.Fill>

</Rectangle>

<Ellipse x:Name="Pointer" Fill="Green" Width="10" Height="10" Canvas.Left="-10" Canvas.Top="-10"/>

<TextBlock Text="Desired Distance: " Canvas.Top="10" Canvas.Right="10" x:Name="DesiredDDisplay"/>

<TextBlock Text="Desired Angle: " Canvas.Top="30" Canvas.Right="10" x:Name="DesiredThetaDisplay"/>

<TextBlock Text="Current Angle: " Canvas.Top="50" Canvas.Right="10" x:Name="CurrentThetaDisplay"/>

<TextBlock Text="Theta Velocity: " Canvas.Top="100" Canvas.Right="10" x:Name="ThetaVelDisplay"/>

<TextBlock Text="Theta Acceleration: " Canvas.Top="120" Canvas.Right="10" x:Name="ThetaAccelerationDisplay"/>

<TextBlock Text="Distance Velocity: " Canvas.Top="140" Canvas.Right="10" x:Name="DistanceVelocityDisplay"/>

<TextBlock Text="Distance Acceleration: " Canvas.Top="160" Canvas.Right="10" x:Name="DistanceAccelerationDisplay"/>

</Canvas>

<Grid Grid.Column="1" Background="DarkGray">

<Grid.RowDefinitions>

<RowDefinition Height="50"/>

<RowDefinition Height="20"/>

<RowDefinition Height="330"/>

<RowDefinition Height="50"/>

</Grid.RowDefinitions>

<Button x:Name="PidActive" Content="PID Active?" Grid.Row="0" Click="PidActive\_Click"/>

<TextBlock x:Name="PidActiveDisplay" Text="False" Grid.Row="1" HorizontalAlignment="Center" Foreground="Red"/>

<Button x:Name="GoBack" Content="Go Back" Grid.Row="4" Click="GoBack\_Click" />

<Grid Grid.Row="2">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="20"/>

<RowDefinition Height="\*"/>

<RowDefinition Height="20"/>

<RowDefinition Height="20"/>

</Grid.RowDefinitions>

<TextBlock Text="P" HorizontalAlignment="Center" Grid.Row="0" Grid.Column="0"/>

<TextBlock Text="I" HorizontalAlignment="Center" Grid.Row="0" Grid.Column="1"/>

<TextBlock Text="D" HorizontalAlignment="Center" Grid.Row="0" Grid.Column="2"/>

<Slider x:Name="PSlider" Grid.Row="1" Grid.Column="0" Orientation="Vertical" HorizontalAlignment="Center" Maximum="10" Minimum="0" ValueChanged="PSlider\_ValueChanged" TickFrequency="0.1" SmallChange="0.1" IsSnapToTickEnabled="True"/>

<Slider x:Name="ISlider" Grid.Row="1" Grid.Column="1" Orientation="Vertical" HorizontalAlignment="Center" Maximum="10" Minimum="0" ValueChanged="ISlider\_ValueChanged" TickFrequency="0.1" SmallChange="0.1" IsSnapToTickEnabled="True"/>

<Slider x:Name="DSlider" Grid.Row="1" Grid.Column="2" Orientation="Vertical" HorizontalAlignment="Center" Maximum="10" Minimum="0" ValueChanged="DSlider\_ValueChanged" TickFrequency="0.1" SmallChange="0.1" IsSnapToTickEnabled="True"/>

<TextBlock x:Name="PValue" Grid.Row="2" Grid.Column="0" Text="0" HorizontalAlignment="Center"/>

<TextBlock x:Name="IValue" Grid.Row="2" Grid.Column="1" Text="0" HorizontalAlignment="Center"/>

<TextBlock x:Name="DValue" Grid.Row="2" Grid.Column="2" Text="0" HorizontalAlignment="Center"/>

</Grid>

</Grid>

</Grid>

</local:GenericModel>

## VelocityModel.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Windows.Threading;

using static System.Net.Mime.MediaTypeNames;

namespace NavMenuNew

{

/// <summary>

/// Interaction logic for Model3.xaml

/// </summary>

public partial class Model3 : GenericModel

{

//define variables

PID.PID anglePID = new PID.PID(500, 0.1);

PID.PID distancePID = new PID.PID(4000, 0.5);

public double currentX = 100;

public double currentY = 100;

public double desiredX = -10;

public double desiredY = -10;

public Model3()

{

InitializeComponent();

// Create the timer event

timer = new DispatcherTimer();

pidTiming = Period / 1000F;

timer.Interval = new TimeSpan(0, 0, 0, 0, Period);

timer.Tick += TimerEvent;

timer.Start();

}

private void TimerEvent(object sender, EventArgs e)

{

// Calculate desiredD and desiredTheta from desiredX and desiredY

double deltaX = desiredX - currentX;

double deltaY = desiredY - currentY;

desiredD = Math.Sqrt(deltaX \* deltaX + deltaY \* deltaY);

if (deltaY >= 0)

{

desiredTheta = Math.PI - Math.Atan(deltaX / deltaY);

}

else

{

if (deltaX >= 0)

{

desiredTheta = -Math.Atan(deltaX / deltaY);

}

else

{

desiredTheta = 2 \* Math.PI - Math.Atan(deltaX / deltaY);

}

}

// Calculate PID

// set DAcc and TAcc to 0 if !pidActive

if (pidActive)

{

// DAcc

DAcc = distancePID.next(desiredD, 0, kP, kI, kD, pidTiming);

//TAcc

if (desiredD < 1)

{

currentX = desiredX;

currentY = desiredY;

DAcc = 0;

DVel = 0;

desiredTheta = 0.5\*Math.PI;

}

// edge cases for crossing 0

if (0 <= currentTheta && currentTheta <= 0.5 \* Math.PI && 1.5 \* Math.PI <= desiredTheta && desiredTheta <= 2 \* Math.PI)

{

TAcc = anglePID.next(desiredTheta, currentTheta + 2 \* Math.PI, kP, kI, kD, pidTiming);

}

else if (0 <= desiredTheta && desiredTheta <= 0.5 \* Math.PI && 1.5 \* Math.PI <= currentTheta && currentTheta <= 2 \* Math.PI)

{

TAcc = anglePID.next(desiredTheta + 2 \* Math.PI, currentTheta, kP, kI, kD, pidTiming);

}

else

{

TAcc = anglePID.next(desiredTheta, currentTheta, kP, kI, kD, pidTiming);

}

}

else

{

DAcc = 0;

TAcc= 0;

}

// Apply TAcc

// Cap at 1/20 pi

if (TVel >= 0)

{

TVel = Math.Min(TVel + TAcc, Math.PI / 20);

}

else

{

TVel = Math.Max(TVel + TAcc, -Math.PI / 10);

}

// Apply DAcc

// always positive

// Cap at 2

DVel = Math.Min(DVel + DAcc, 2);

// Apply TVel

currentTheta = (currentTheta + TVel)%(2\*Math.PI);

while(currentTheta < 0)

{

currentTheta += 2\*Math.PI;

}

// Apply DVel

if (currentTheta >= 1.5\*Math.PI) // Apply North - West

{

currentX -= DVel\*Math.Cos(currentTheta-1.5\*Math.PI);

currentY -= DVel\*Math.Sin(currentTheta-1.5\*Math.PI);

}

else if(currentTheta >= Math.PI) // Apply South - West

{

currentX -= DVel \* Math.Sin(currentTheta - Math.PI);

currentY += DVel \* Math.Cos(currentTheta - Math.PI);

}

else if (currentTheta >= 0.5 \* Math.PI) // Apply South - East

{

currentX += DVel \* Math.Cos(currentTheta - 0.5 \* Math.PI);

currentY += DVel \* Math.Sin(currentTheta - 0.5 \* Math.PI);

}

else // Apply North - East

{

currentX += DVel \* Math.Cos(currentTheta);

currentY -= DVel \* Math.Sin(currentTheta);

}

// Apply Resistance

TVel \*= 0.98;

if (TVel > 0.25)

{

TVel = 0.25; // Maximum

}

else if (TVel < -0.25)

{

TVel = -0.25; // -Maximum

}

else if(-0.001 < TVel && TVel < 0.001)

{

TVel = 0; // Approaching Zero

}

if (DVel > 0.01)

{

DVel = 0.98 \* DVel;

}

else

{

DVel = 0; // Approaching Zero

}

// update Visuals

Canvas.SetLeft(Car, currentX-50);

Canvas.SetTop(Car, currentY-31.25);

// Rotate the Car

RotateTransform carRotateTransform = new RotateTransform(360 \* currentTheta / (2 \* Math.PI) - 90);

Car.RenderTransform = carRotateTransform;

DesiredDDisplay.Text = "Desired Distance: " + Math.Round(desiredD,2);

DesiredThetaDisplay.Text = "Desired Angle: " + Math.Round(desiredTheta/Math.PI,2) + " Pi";

CurrentThetaDisplay.Text = "Current Angle: " + Math.Round(currentTheta/Math.PI,2) +" Pi";

ThetaVelDisplay.Text = "Theta Velocity: " + Math.Round(TVel,7);

ThetaAccelerationDisplay.Text = "Theta Acceleration: " + Math.Round(TAcc,7);

DistanceVelocityDisplay.Text = "Distance Velocity: " + Math.Round(DVel,2);

DistanceAccelerationDisplay.Text = "Distance Acceleration: " + Math.Round(DAcc,2);

}

private void PSlider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

// P Changed

kP = Math.Round(Convert.ToDouble(PSlider.Value), 2);

PValue.Text = Convert.ToString(kP);

}

private void ISlider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

// I Changed

kI = Math.Round(Convert.ToDouble(ISlider.Value), 2);

IValue.Text = Convert.ToString(kI);

}

private void DSlider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

// D Changed

kD = Math.Round(Convert.ToDouble(DSlider.Value), 2);

DValue.Text = Convert.ToString(kD);

}

private void PidActive\_Click(object sender, RoutedEventArgs e)

{

// Flip pidActive

pidActive = !pidActive;

PidActiveDisplay.Text = Convert.ToString(pidActive);

if (pidActive) // Update UI

{

PidActiveDisplay.Foreground = new SolidColorBrush(Color.FromRgb(0, 185, 0));

anglePID.It = 0;

distancePID.It = 0;

}

else if (!pidActive)

{

PidActiveDisplay.Foreground = new SolidColorBrush(Color.FromRgb(255, 0, 0));

}

}

private void GoBack\_Click(object sender, RoutedEventArgs e)

{

// Go back to NavMenu

UserControl Model = new NavMenu();

Canvas.SetLeft(Model, 0);

Canvas.SetTop(Model, 0);

Canvas MainCanvas = (Canvas)this.Parent;

MainCanvas.Children.Clear();

MainCanvas.Children.Add(Model);

}

private void largeCanvas\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

// Set Desired X/Y

Point mousePos = e.GetPosition(largeCanvas);

desiredX = mousePos.X + 5;

desiredY = mousePos.Y + 5;

Canvas.SetLeft(Pointer, desiredX);

Canvas.SetTop(Pointer, desiredY);

}

}

}

# Systems Testing

## Testing Against Objectives

During the Requirements Analysis, I established a list of SMART objectives for me to meet by the end of this project. This was the list:

1. Interactive and intuitive user interface (A,R)
2. Contains a control panel to interact with the PID model (S,M,A,R)
3. Includes feedback of calculations and current values (M,A,R)
4. Has expandable menus detailing the processes behind each part of PID (S,M,A,R)
5. Three detailed models should be available (M,A,R)
6. Rotation, extension, and velocity control should be demonstrated (S,M,A,R)
7. Relationships should aim to be proportional or polynomial (A,R)
8. Must be light enough to run on light computer systems, running windows (S,M,A,R)
9. Must be finished before Spring exams 2023 (S,M,A,R,T)

For each of these, I ran at least one test to show whether I have successfully achieved that objective or not.

|  |  |  |  |
| --- | --- | --- | --- |
| Objective | Test | Result | Achieved? |
| Interactive and intuitive user interface | Have another person try use the program with minimal input | They weren’t keen to read the block of text at the start, and some functions like setting the desired position aren’t clear. | The UI functioned, but some elements weren’t clear |
| Contains a control panel to interact with the PID model | I should be able to Activate and Deactivate the PID algorithm, editing its parameters from the UI | The UI sliders and activation button works exactly as intended |  |
| Includes feedback of calculations and current values | Can I tell what is happening throughout the program running? | I am told the calculations on the first screen, and there is a constant feedback as values change |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Has expandable menus detailing the processes behind each part of PID | Can I select a part of the explanation to get it in more detail? | | | There is a large detailed explanation on the first page, but all the detail is shown in one go. | All the info is there, but it’s a bit much to read |
| Three detailed models should be available | Can I move between three models and the main menu? | | | https://youtu.be/\_xPzq03zfMc |  |
| Rotation, extension, and velocity control should be demonstrated | Demonstrate each of these 3 principles in the models. | https://youtu.be/MDJmmLw7QXw | | |  |
| Relationships should aim to be proportional or polynomial | Show how each calculation has been simplified |  | | |  |
| Must be light enough to run on light computer systems, running windows | Try run it on a light system. How smoothly | | https://youtu.be/MDJmmLw7QXw | |  |
| Must be finished before Spring exams 2023 | Was it finished before the exams? | | Yes, The program was finished by 16/12/2022 | |  |

## Testing Handling Exceptions

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| M# | P | I | D | PID? | Expectation | Result |
| 1 | 0 | 0 | 0 | True | Nothing Happens | <https://youtu.be/Znvp0qAr2xA> |
| 2 | 0 | 0 | 0 | True | Nothing Happens | <https://youtu.be/Znvp0qAr2xA?t=14> |
| 3 | 0 | 0 | 0 | True | Nothing Happens | <https://youtu.be/Znvp0qAr2xA?t=25> |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 5 | 5 | 5 | False | Nothing Happens | <https://youtu.be/Znvp0qAr2xA?t=36> |
| 2 | 5 | 5 | 5 | False | Nothing Happens | <https://youtu.be/Znvp0qAr2xA?t=58> |
| 3 | 5 | 5 | 5 | False | Nothing Happens | <https://youtu.be/Znvp0qAr2xA?t=70> |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 5 | 0 | 0 | True | The angle will be met with fluctuations | <https://youtu.be/Znvp0qAr2xA?t=83> |
| 2 | 5 | 0 | 0 | True | The point will be met with fluctuations | <https://youtu.be/Znvp0qAr2xA?t=101> |
| 3 | 5 | 0 | 0 | True | The point will be met with fluctuations | <https://youtu.be/Znvp0qAr2xA?t=114> |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 5 | 0 | True | The angle will accelerate uncontrollably | <https://youtu.be/Znvp0qAr2xA?t=140> |
| 2 | 0 | 5 | 0 | True | The angle and length will accelerate uncontrollably | <https://youtu.be/Znvp0qAr2xA?t=155> |
| 3 | 0 | 5 | 0 | True | The angle and speed will accelerate uncontrollably | <https://youtu.be/Znvp0qAr2xA?t=170> |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 5 | True | Nothing Happens | <https://youtu.be/Znvp0qAr2xA?t=194> |
| 2 | 0 | 0 | 5 | True | Nothing Happens | <https://youtu.be/Znvp0qAr2xA?t=204> |
| 3 | 0 | 0 | 5 | True | Nothing Happens | <https://youtu.be/Znvp0qAr2xA?t=214> |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 5->0 | 0 | 0 | True | Angle fluctuates, comes to a smooth stop | <https://youtu.be/Znvp0qAr2xA?t=221> |
| 2 | 5->0 | 0 | 0 | True | Arm fluctuates, comes to a smooth stop | <https://youtu.be/Znvp0qAr2xA?t=235> |
| 3 | 5->0 | 0 | 0 | True | Velocity fluctuates, comes to a smooth stop | <https://youtu.be/Znvp0qAr2xA?t=251> |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 5->0 | 0 | True | Rapid acceleration, into smooth stop | <https://youtu.be/Znvp0qAr2xA?t=266> Test 19\* | These three tests proved unsuccessful. Once the constant was at 0, the velocity didn’t slow to 0. This is because although kI is 0, the It variable was still positive. |
| 2 | 0 | 5->0 | 0 | True | Rapid acceleration, into smooth stop | <https://youtu.be/Znvp0qAr2xA?t=286> |
| 3 | 0 | 5->0 | 0 | True | Rapid acceleration, into smooth stop | <https://youtu.be/Znvp0qAr2xA?t=308> |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 5->0 | True | Nothing | <https://youtu.be/Znvp0qAr2xA?t=327> |
| 2 | 0 | 0 | 5->0 | True | Nothing | <https://youtu.be/Znvp0qAr2xA?t=340> |
| 3 | 0 | 0 | 5->0 | True | Nothing | <https://youtu.be/Znvp0qAr2xA?t=355> |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M# | ~Angle From | ~Angle To | Expectation | Result |
| 2 | 0 < θ < | 0 < θ < | Minimal Movement | <https://youtu.be/Znvp0qAr2xA?t=370> |
| 2 | 0 < θ < | < θ<π | Moves Clockwise by | <https://youtu.be/Znvp0qAr2xA?t=384> |
| 2 | 0 < θ < | π < θ < | Moves either way by π | <https://youtu.be/Znvp0qAr2xA?t=396> |
| 2 | 0 < θ < | < θ < 2π | Moves Anticlockwise by through 0 | <https://youtu.be/Znvp0qAr2xA?t=408> |
| 2 | < θ<π | 0 < θ < | Moves Anticlockwise by | <https://youtu.be/Znvp0qAr2xA?t=420> |
| 2 | < θ<π | < θ<π | Minimal Movement | <https://youtu.be/Znvp0qAr2xA?t=431> |
| 2 | < θ<π | π < θ < | Moves Clockwise by | <https://youtu.be/Znvp0qAr2xA?t=443> |
| 2 | < θ<π | < θ < 2π | Moves either way by π | <https://youtu.be/Znvp0qAr2xA?t=456> |
| 2 | π < θ < | 0 < θ < | Moves either way by π | <https://youtu.be/Znvp0qAr2xA?t=470> |
| 2 | π < θ < | < θ<π | Moves Anticlockwise by | <https://youtu.be/Znvp0qAr2xA?t=481> |
| 2 | π < θ < | π < θ < | Minimal Movement | <https://youtu.be/Znvp0qAr2xA?t=494> |
| 2 | π < θ < | < θ < 2π | Moves Clockwise by | <https://youtu.be/Znvp0qAr2xA?t=504> |
| 2 | < θ < 2π | 0 < θ < | Moves Clockwise by through 0 | <https://youtu.be/Znvp0qAr2xA?t=521> |
| 2 | < θ < 2π | < θ<π | Moves either way by π | <https://youtu.be/Znvp0qAr2xA?t=533> |
| 2 | < θ < 2π | π < θ < | Moves Anticlockwise by | <https://youtu.be/Znvp0qAr2xA?t=548> |
| 2 | < θ < 2π | < θ < 2π | Minimal Movement | <https://youtu.be/Znvp0qAr2xA?t=567> |

Model 3 uses the same rotary logic as Model 2 so these tests are only required once.

# Evaluation

## Post Production Interviews

### Sponsor Interview

**Me:** What is your overall opinions on the finished program?

***Mr Maher:*** *While the UI and teaching elements certainly could be improved to be clearer, the actual models and algorithms of the program very impressive.*

**Me:** Do you think the program explains the uses for Differentiation and Integration well?

***Mr Maher:*** *Not particularly. The explanation on the main menu feels tacked on. It is not very dynamic or explanatory of what Integration or Differentiation are and how to do them.*

**Me:** Did you find yourself engaged with what was happening?

***Mr Maher:*** *Yes, every motion was very smooth and all of the models were interesting and different*

**Me:** Was the program suitably interactive and clear to understand for teaching?

***Mr Maher:*** *It was certainly interactive, and would keep the students attention, however I feel it doesn’t do a good job of actually teaching the maths to the students*

**Me:** Do you think this would be helpful as an automatic control unit?

***Mr Maher:*** *With the right constants programmed in, the program works amazingly for automatic control. Don’t get me wrong, Tesla wont be using it for their cars any time soon, but for the time frame and level of expertise you had, this is very impressive.*

### End-User Interviews

#### Interview with Alex:

**Me:** What is your overall opinions on the finished program?

***Alex:*** *It is a very good demonstration of PID and allows me to test the algorithm in many different models which display some of possible applications of PID.*

**Me:** Do you think the program explains the uses for Differentiation and Integration well?

***Alex:*** *yes, the program explains how PID works without being too technical.*

**Me:** Did you find yourself engaged with what was happening?

***Alex:*** *yes, the program requires engagement for it to actually work - without any engagement the program does not change.*

**Me:** Was the program suitably interactive and clear to understand?

***Alex:*** *yes it comes with clear instructions to use the program, and it is very interactive, as for the program to actually be used I have to put inputs to change graphical elements, whether it’s clicking on the screen to direct a car or using keyboard inputs to move a rectangle.*

**Me:** Do you think this tool would help more people where you struggled?

***Alex:*** *I don’t think that it would help that much, it helps show how integration and how differentiation can be used in different applications, but learning how it works is separate to this.*

#### Interview with Jakub:

**Me:** What is your overall opinions on the finished program?

***Jakub:*** *It was good, it was interesting. Intriguing to mess about with the sliders, and see how it affected what was going on.*

**Me:** Do you think the program explains the uses for Differentiation and Integration well?

***Jakub:*** *Yes. It explains clearly where and how they are being used, and then demonstrates them well.*

**Me:** Did you find yourself engaged with what was happening?

***Jakub:*** *Yeah, there was a lot of moving parts, so it keeps you looking, especially with the pointers and the extra statistics, of the velocity and such on the side.*

**Me:** Was the program suitably interactive and clear to understand?

***Jakub:*** *Yeah, there wasn’t quite as much of a window into what was happening as I hoped there might be, but you could still tell what was going on. As for the controls, they were intuitive, but unfortunately a bit unclear. Like for the first model, having to select the button to be able to rotate the rectangle seemed really strange – but as I saw the Sliders were adjusting I understood it wasn’t entirely the fault of the programming and more of the system.*

**Me:** Do you think this tool would help more people where you struggled?

***Jakub:*** *Some people, I think it would heavily depend on what type of learner they are. Some would be so fascinated with it, and want to go out and learn how it works, but for some people it could just as easily confuse them more.*

## Summary

Overall, while the program does not function great as an actual teaching tool, it does a fantastic job of demonstrating how the proportional, integral and differential relationships can be used and can very much grab the users attention and hold it.

In a classroom, it may not be the best tool to get students using themselves, but for a brief demonstration on how these relationships are usable in real life, and making certain students curious to how it works, this program projected onto a board at the front of the class would work wonderfully.

As for the PID module, there is only a minor change necessary for it to be entirely functional and useful for not just this program, but other programs as well. As it is completely self-contained, it can be reused in separate contexts extremely easily.

The program also greatly demonstrates the wide range of contexts that the algorithm can be applied to. This could lead to inspiring students to look into autonomous control systems, and potentially plant the seed of an interest for either a hobby or career in robotics, AI or control systems.

In conclusion, while I may not have met my original objectives to be an effective teaching tool, I have still met my other objectives, and as the original brief was quite vague on what this tool could be used for, this is still extremely successful.