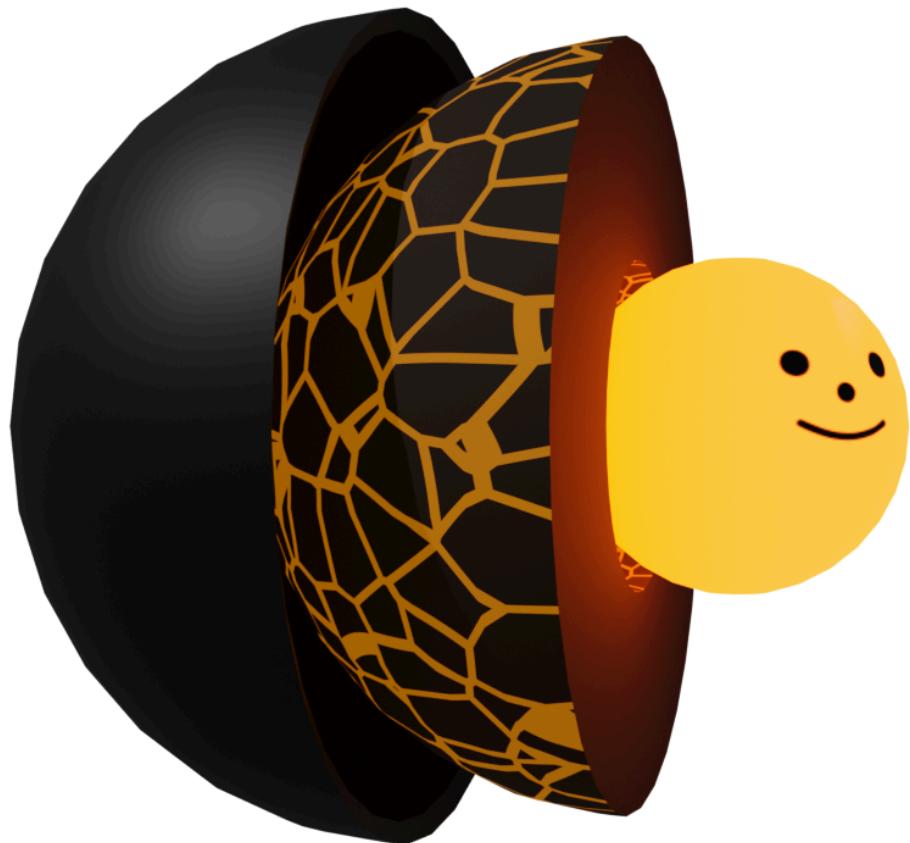


# RIT VEXU Software Engineering Notebook

2023-2024



# Software Development Process

- Subrepo - why we use it, why tried to switch to submodules, and why we switched back
- Github project board
- Wiki
- ~~Clang tidy~~
- ~~Github Actions~~

## Github Actions

This year, our team enhanced our workflow by integrating GitHub Actions into our software development process. One notable addition was an action to build our C/C++ code in the appropriate Vex environment. This automated process involves a series of steps, including checking out the repository, downloading and unzipping the Vex Robotics SDK and toolchain, and compiling the code using Makefile. A key feature of this GitHub Action is its ability to send a Slack notification to our team channel whenever a build fails, ensuring prompt awareness and response. Furthermore, it helps maintain code integrity by preventing the merging of pull requests with failing builds. This complements our other GitHub Action for building Doxygen documentation and deploying it to GitHub Pages, allowing for seamless documentation and code management. This systematic approach aligns with our commitment to maintaining a neat, organized, and efficient engineering process.



Continuous Integration directly improves the quality of our code.

A screenshot of the OdometryBase Class Reference documentation. It shows the inheritance diagram for OdometryBase, which is an abstract class. It inherits from Odometry3Wheel and OdometryTank. Below the diagram, there is a section for "Public Member Functions".

```
#include <odometry_base.h>
```

Inheritance diagram for OdometryBase:

```
graph TD; OdometryBase[OdometryBase] --> Odometry3Wheel[Odometry3Wheel]; OdometryBase --> OdometryTank[OdometryTank]
```

Public Member Functions

Automatically generated documentation.

## Auto-Notebook

Alongside the automatic automatic documentation, whenever Core is updated or we manually trigger it, a Github Action copies the reference manual, exports the most up to date version of our written notebook document, stitches them together and deploys to a webpage. This is publicly available for any person wishing to see our software development process. The most valuable effect, though, is automating most of the formatting work for our notebook that used to require a team member to use valuable pre-competition time to sit down, append, format and export the notebook.

## Clang-Tidy

In an effort to improve the quality, reduce headaches, and overall make our code easier to read, write, and understand, we enabled many more warnings than what is supplied with the default Vex project Makefile. These warnings deal with uninitialized variables, missing returns, and other simple code errors that nonetheless have the tendency to introduce tiny, hard to track down bugs. However, sometimes these warnings do not explore deep enough and another tool must be used. We integrated clang-tidy, a c++ linter developed by the clang compiler project, to inspect our code. With a simple switch of a variable in the Makefile, we run clang-tidy during builds which gives many insights into the code that plain compiler warnings do not. Though it does increase compilation times, it tells us about code that is bug prone, poor for performance, and many other checks developed and validated by the wider C++ community.

# Core: Fundamentals

- Odometry
  - Tank (same old)
  - GPS + Odometry
- Drivetrain
  - Tank Drive
    - What it do
      - DriveForward, TurnToHeading
      - DriveToPoint
    - Whats new
      - Pure pursuit
      - Brake mode
- Control systems
  - What it do
    - FeedbackBase (Swapping control loops)
    - PID
    - PIDFF
    - Bang Bang?
    - Take Back Half?
    - Motion Profile
  - Whats new
    - Trapezoid with velocity and such
- ACS
  - What it do
    - Organizing commands in orders, not having 1000 if statements
  - Whats new
    - Shortcuts
    - Memory Management that isn't bad
    - Cyborg control
      - Using auto paths in teleop after we convince the drivers not to be luddites
- Logger/Serializer
- Screen Stuff
  - Graphing
  - Auto Choosing
  - Odometrying
  - Motor stats
  - Pid Tuner
  - Widgets
  - Not redeploying code every 30 seconds

- Catapult Shenanigans
  - Sequencing, deploying
- Vision (to be written later)
  - Localizing nuts
  - Squirrel behavior
-

# Control Loops

In order for the Autonomous Command Structure to function, we need a way to tell the robot how we want it to move. There are two broad categories of telling a robot to achieve a requested position - Feedback and Feedforward. Feedback relies on sensors and adjusts the output of the robot according to the error between where it is and where it wants to be. On the other hand, a feedforward controller takes a mathematical model of the system and creates outputs based on what it calculates to be the necessary output to achieve the goal. Additionally, there are simpler methods like Bang-Bang or Take Back Half. These adjust the outputs based on the current position relative to the target, where Take Back Half gradually refines the output until it settles at the desired position. These controller types work for many applications, but a combination of them can achieve an even better control over robot actuators.

## PID

A PID controller is perhaps the most common type of Feedback control. It uses measurements of the error at its current state (proportional), measurements of how the error was in the past (integral) and measurements of how the error changes over time(derivative). The controller acts accordingly to bring the errors towards 0. We implemented a standard PID controller but made some alterations to fit our needs. The most important of these are custom error calculations. The standard error calculation function (*target - measured*) works for many of our uses but causes problems when we use a PID controller to control angles. Since angles wrap around at 360 degrees or  $2\pi$  radians we wrote our own error calculation function that gives the error that accounts for this wrapping.

## Feedforward

A feedforward controller differs from a feedback controller in that it does not rely on any measurement of error to command a system. Instead, built into a feedforward controller is a mathematical model of the domain. When a target is requested by the controller, the model is queried to figure out what the robot actuators must output to achieve that target. A key advantage of this form of control is that instead of waiting for an error to build up in the system, the controller acts directly to achieve the target and can reach the target much faster.

## Bang-Bang

Bang-Bang control is a straightforward control methodology where the output to the system is either fully on or fully off, with no intermediate states. It's used for systems where fine control isn't necessary or possible. In this method, when the process variable is below the setpoint, the controller output is set to maximum; when above, it's set to minimum. This approach is simple and often used for systems with high inertia or where the precise control of the variable isn't critical. However, it can lead to oscillations around the setpoint and isn't suited for systems requiring precise regulation.

## Take Back Half (TBH)

The Take Back Half (TBH) method is an iterative approach used to refine control in systems where overshoot is a concern. This method adjusts the output by taking back half the value of the output each time the controlled variable overshoots the target. The adjustment continues until the system settles close to the desired setpoint. TBH is particularly useful in scenarios where a fine balance between responsiveness and stability is needed, as it reduces the oscillation or overshoot often seen in simpler control methods. It's a practical choice for systems where a PID controller might be too complex or unnecessary. TBH controllers only have one tuning parameter which allows for an incredibly easy tuning experience.

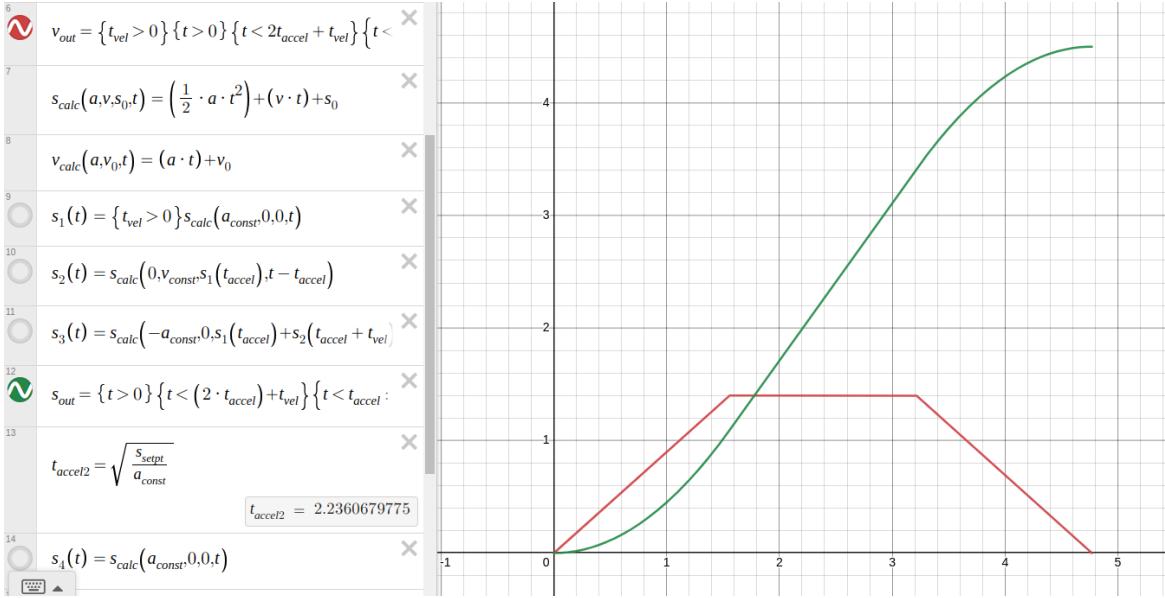
## Generic Feedback

Different control systems work best in different environments. Because of this, we found ourselves switching control schemes often enough that rewriting the code each time was time consuming and often led to rushed, worse quality code. To solve this problem we implemented a generic feedback interface so that none of our subsystem code needs to change when we use a different control scheme. Instead, the subsystem reports to the controller where it wants to be, measurements from its environment and some information about the system's capabilities and the controller will report back the actions needed to achieve that target. This allows for much faster prototyping and cleaner, less tightly coupled code.

## Motion Profile

As we learn from each event, our team has evolved our approach to robot control systems, transitioning from a simple PID controller to a more sophisticated Motion Profile controller. The PID system, while fundamental, had its drawbacks, such as limited speed specification, poor response to wheel slipping, and slower reaction times. These limitations highlighted the need for a more advanced control mechanism.

Our Motion Profile controller represents a significant upgrade. It integrates precise control over position, acceleration, and velocity, allowing for optimized performance of our robot's subsystems. Unlike the PID controller, which reacts only to discrepancies between actual and desired states, our Motion Profile controller proactively manages the robot's movements. It anticipates the required actions, thereby reducing response lags. Moreover, it avoids the rigidness of a pure feedforward controller by adapting dynamically to changing conditions in competition scenarios.



A key feature of the Motion Profile controller is its ability to handle varying accelerations. This functionality enables our robot to accelerate efficiently without wheel slipping, always maintaining optimal acceleration. This year, we've further refined our Motion Profile to accommodate non-zero starting and ending velocities. This enhancement allows for the seamless chaining of complex movements, ensuring smoother transitions and more fluid motion during competition tasks.

# Auto Command Structure (ACS)

## Principle

A recent addition to our core API was that of the Autonomous Command Structure. No more will our eyes glaze over staring at brackets as we trawl through an ocean of nameless functions nor lose our way in a labyrinthine state machine constructed not of brick and stone but blocks of ifs and whiles. Instead, we provide named Commands for all the actions that our robot can execute and infrastructure to run them sequentially or concurrently. The API is written in a declarative way allowing even programmers unfamiliar with the code to see a step by step, annotated guide to our autonomous path while keeping the procedures of how to execute the actions from hurting the readability of the path.

```
CommandController auto_non_loader_side(){
    int non_loader_side_full_court_shot_rpm = 3000;
    CommandController non_loader_side_auto;

    non_loader_side_auto.add(new SpinRPMCommand(flywheel_sys, non_loader_side_full_court_shot_rpm));
    non_loader_side_auto.add(new WaitUntilUpToSpeedCommand(flywheel_sys, 10));
    non_loader_side_auto.add(new ShootCommand(intake, 2));
    non_loader_side_auto.add(new FlywheelStopCommand(flywheel_sys));
    non_loader_side_auto.add(new TurnDegreesCommand(drive_sys, turn_fast_mprofile, -60, 1));
    non_loader_side_auto.add(new DriveForwardCommand(drive_sys, drive_fast_mprofile, 20, fwd, 1));
    non_loader_side_auto.add(new TurnDegreesCommand(drive_sys, turn_fast_mprofile, -90, 1));
    non_loader_side_auto.add(new DriveForwardCommand(drive_sys, drive_fast_mprofile, 2, fwd, 1));
    non_loader_side_auto.add(new SpinRollerCommand(roller));

    return non_loader_side_auto;
}
```

ACS code from the 2023 competition season

## Updates

This season, we found ourselves annoyed with having to repeat basic things such as path.add(...) and having to write new ThingCommand over and over again. Our first solution to this was “shortcuts”. These were member functions of subsystems that would allocate, initialize and return an auto command for that subsystem. So, instead of path.add(new DriveForwardCommand(drive\_sys, drive\_fast\_mprofile, 20, fwd)) we could simply write path.add(drive\_sys.DriveForwardCommand(20, fwd)). This reduced a great deal of typing but still left us with some issues.

The most hazardous, rather than simply annoying downside of last year's system, was the memory unsafety of this system. Since our auto commands must use virtual functions, they must be on the other end of a pointer. So, they must be allocated using new or they must be initialized statically before we write the path which is a terrible user experience (Though, if constrained by an embedded system where allocating on the heap was deemed dangerous, the system could work with this). This became a real issue when we began to write more complicated constructs such as branching, asynchronous, and repeated commands as it became dangerously unclear who was responsible for deallocating these objects. As a solution for this, we developed an RAII wrapper for the Auto Command Interface. Inspired by C++'s std::unique\_ptr, this wrapper provides a memory safe, value based way of using auto

commands while still maintaining their adaptability. We used C++'s ideas of move semantics and 'Resource Allocation Is Initialization' to practically solve memory management so programmers (and even non programmers) can focus on writing paths.

```
CommandController cmd{
    odom.SetPositionCmd({.x = 16.0, .y = 16.0, .rot = 225}),
    // 1 - Turn and shoot preload
    {
        cata_sys.Fire(),
        drive_sys.DriveForwardCmd(dist, REV),
        DelayCommand(300),
        cata_sys.StopFiring(),
        cata_sys.IntakeFully(),
    },
    // 2 - Turn to matchload zone & begin matchloading
    drive_sys.DriveForwardCmd(dist + 2, FWD, 0.5)
        .with_timeout(1.5),

    // Matchloading phase
    Repeat{
        odom.SetPositionCmd({.x = 16.0, .y = 16.0, .rot = 225}),
        intakeToCata.with_timeout(1.75),
        cata_sys.Fire(),
        drive_sys.DriveForwardCmd(10, REV, 0.5),
        cata_sys.StopFiring(),
        cata_sys.IntakeFully(),
        drive_sys.TurnToHeadingCmd(load_angle, 0.5),
        drive_sys.DriveForwardCmd(12, FWD, 0.2).with_timeout(1.7),
    }.until(TimeSinceStartExceeds(30))
};
```

ACS code going into the 2024 competition season

Now that we were free to use auto commands without fear for leaking memory or messing with currently running commands, we began to create more powerful constructs such as branching on runtime information, timeouts so the robot can decide what to do based on how much time is left in the auto or skills period, fearless concurrency (driving and reloading at the same time), and a much much nicer user interface. This declarative, safe, and straightforward method of writing auto paths lets us spend less time writing and debugging custom code and more time exploring and optimizing auto paths.

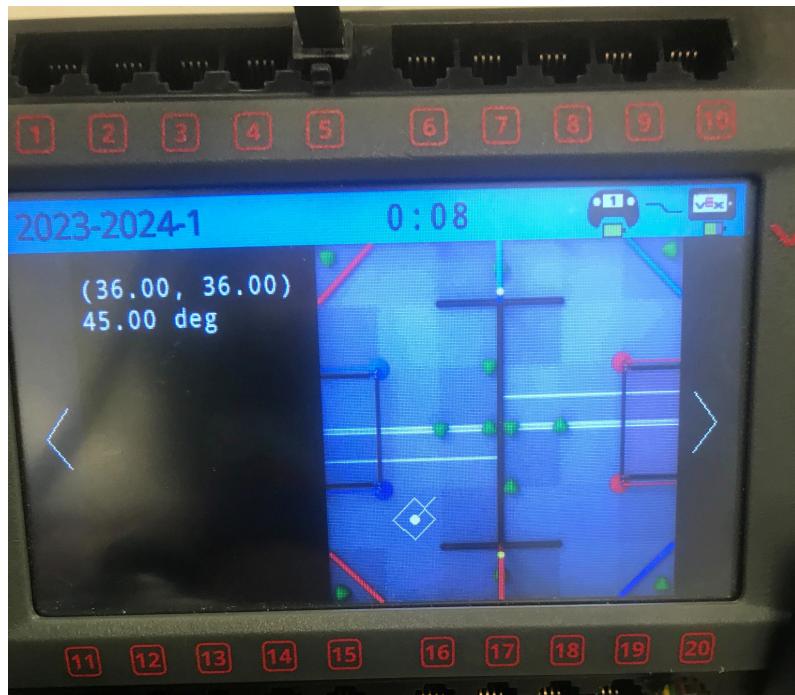
## Serializer

One pain point we found last year was configuring auto paths, color targets, path timeouts, and other parameters that changed match to match but that also should be persistent. Commonly, we found ourselves redeploying code at the last minute before a match. To solve this, we wrote a class that takes control of a file on the SD card to which users can read and

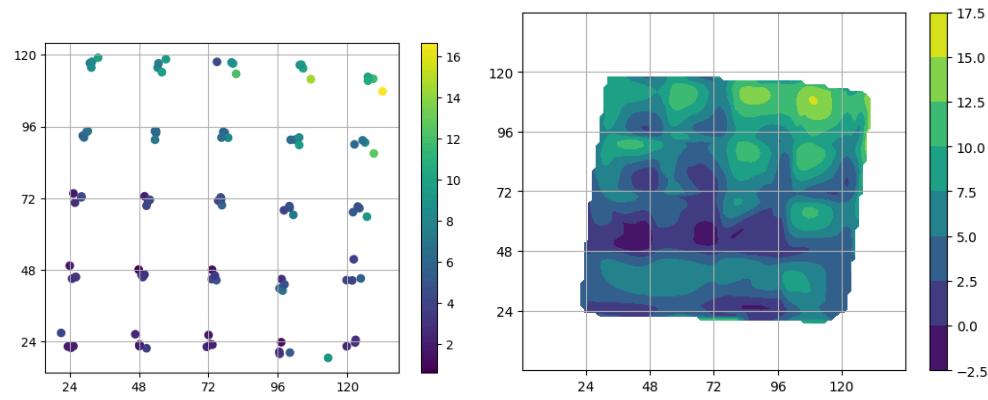
write values at runtime. This keeps us from having to change a value, redeploy, repeat which cost us valuable time in the past.

Draft work for previous section

- Cata system
- Biggerer and betterer screen system
  - Buttons and sliders enabling hardware to test speeds without a coder redeploying every 5 seconds
  - PID, PIDFF tuners
  - Odometry Map



- 
- Motor stats
- Auto chooser
- Cata State Representation for debugging, tuning
- 
- GPS testing (with all the pretty plots (hell yeah))
- Using GPS to tune odometry



- *These pics were from testing it but i think they show a lot about the drift of our wheeled odom*
- *Also reasonable to point out that the middle was more accurate than the edges. Especially looking at the 24 row (near the wall but not too far off from drift) to 48 row (farther from the wall but still not too far from drift)*
- GPS odom
  - With our fun filter
  - How it fits with our other odometry components
- Cata system state machine
  - Message passing parallelism 😎
- Brake mode stuff
  - Velocity brake
  - Smart brake
  - Motivation. Why pure position brake is bad
- Pure pursuit
  - Idrk ask mcgee
- Failed experiments in IMU fusion odom?
  - We cant buy submarine IMUs :(
- Motion controller coolerness + how it interacts with pure pursuit
- Layout system

# Core: Ongoing Projects

- N pod odom
- Core-rs - big things cooking
  - Bridge layer if theres anything interesting to say about it
- Vex debug board if we get it working
- Vex sim?



## Machine Learning

-

## Core: The Funny

Only maybe, this is not professional or something

- 3d renderer
- Video player
- NES (Non-nintendo emulated system) Emulator (did outreach with this cuz a middle schooler was a big fan)

# RIT VEXU Core API

Generated by Doxygen 1.10.0



---

<b>1 Core</b>	<b>1</b>
1.1 Getting Started . . . . .	1
1.2 Features . . . . .	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy . . . . .	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List . . . . .	5
<b>4 File Index</b>	<b>9</b>
4.1 File List . . . . .	9
<b>5 Class Documentation</b>	<b>11</b>
5.1 AndCondition Class Reference . . . . .	11
5.1.1 Member Function Documentation . . . . .	11
5.1.1.1 test() . . . . .	11
5.2 Async Class Reference . . . . .	12
5.2.1 Detailed Description . . . . .	12
5.2.2 Member Function Documentation . . . . .	13
5.2.2.1 run() . . . . .	13
5.3 AutoChooser Class Reference . . . . .	13
5.3.1 Detailed Description . . . . .	14
5.3.2 Constructor & Destructor Documentation . . . . .	14
5.3.2.1 AutoChooser() . . . . .	14
5.3.3 Member Function Documentation . . . . .	14
5.3.3.1 draw() . . . . .	14
5.3.3.2 get_choice() . . . . .	15
5.3.3.3 update() . . . . .	15
5.3.4 Member Data Documentation . . . . .	15
5.3.4.1 choice . . . . .	15
5.3.4.2 list . . . . .	15
5.4 AutoCommand Class Reference . . . . .	16
5.4.1 Member Function Documentation . . . . .	17
5.4.1.1 on_timeout() . . . . .	17
5.4.1.2 run() . . . . .	17
5.4.2 Member Data Documentation . . . . .	17
5.4.2.1 timeout_seconds . . . . .	17
5.5 BangBang Class Reference . . . . .	18
5.5.1 Member Function Documentation . . . . .	18
5.5.1.1 get() . . . . .	18
5.5.1.2 init() . . . . .	18
5.5.1.3 is_on_target() . . . . .	19
5.5.1.4 set_limits() . . . . .	19

---

5.5.1.5 update()	19
5.6 BasicSolenoidSet Class Reference	20
5.6.1 Detailed Description	20
5.6.2 Constructor & Destructor Documentation	20
5.6.2.1 BasicSolenoidSet()	20
5.6.3 Member Function Documentation	21
5.6.3.1 run()	21
5.7 BasicSpinCommand Class Reference	21
5.7.1 Detailed Description	22
5.7.2 Constructor & Destructor Documentation	22
5.7.2.1 BasicSpinCommand()	22
5.7.3 Member Function Documentation	23
5.7.3.1 run()	23
5.8 BasicStopCommand Class Reference	23
5.8.1 Detailed Description	24
5.8.2 Constructor & Destructor Documentation	24
5.8.2.1 BasicStopCommand()	24
5.8.3 Member Function Documentation	24
5.8.3.1 run()	24
5.9 Branch Class Reference	25
5.9.1 Detailed Description	26
5.9.2 Member Function Documentation	26
5.9.2.1 on_timeout()	26
5.9.2.2 run()	26
5.10 screen::ButtonWidget Class Reference	26
5.10.1 Detailed Description	27
5.10.2 Constructor & Destructor Documentation	27
5.10.2.1 ButtonWidget() [1/2]	27
5.10.2.2 ButtonWidget() [2/2]	27
5.10.3 Member Function Documentation	27
5.10.3.1 update()	27
5.11 CommandController Class Reference	28
5.11.1 Detailed Description	28
5.11.2 Constructor & Destructor Documentation	28
5.11.2.1 CommandController()	28
5.11.3 Member Function Documentation	29
5.11.3.1 add() [1/3]	29
5.11.3.2 add() [2/3]	29
5.11.3.3 add() [3/3]	29
5.11.3.4 add_cancel_func()	31
5.11.3.5 add_delay()	31
5.11.3.6 last_command_timed_out()	31

---

5.11.3.7 run()	32
5.12 Condition Class Reference	32
5.12.1 Detailed Description	32
5.13 CustomEncoder Class Reference	32
5.13.1 Detailed Description	33
5.13.2 Constructor & Destructor Documentation	33
5.13.2.1 CustomEncoder()	33
5.13.3 Member Function Documentation	33
5.13.3.1 position()	33
5.13.3.2 rotation()	34
5.13.3.3 setPosition()	34
5.13.3.4 setRotation()	34
5.13.3.5 velocity()	34
5.14 DelayCommand Class Reference	35
5.14.1 Detailed Description	36
5.14.2 Constructor & Destructor Documentation	36
5.14.2.1 DelayCommand()	36
5.14.3 Member Function Documentation	36
5.14.3.1 run()	36
5.15 DriveForwardCommand Class Reference	36
5.15.1 Detailed Description	37
5.15.2 Constructor & Destructor Documentation	37
5.15.2.1 DriveForwardCommand()	37
5.15.3 Member Function Documentation	38
5.15.3.1 on_timeout()	38
5.15.3.2 run()	38
5.16 DriveStopCommand Class Reference	38
5.16.1 Detailed Description	39
5.16.2 Constructor & Destructor Documentation	39
5.16.2.1 DriveStopCommand()	39
5.16.3 Member Function Documentation	39
5.16.3.1 on_timeout()	39
5.16.3.2 run()	40
5.17 DriveToPointCommand Class Reference	40
5.17.1 Detailed Description	41
5.17.2 Constructor & Destructor Documentation	41
5.17.2.1 DriveToPointCommand() [1/2]	41
5.17.2.2 DriveToPointCommand() [2/2]	41
5.17.3 Member Function Documentation	42
5.17.3.1 run()	42
5.18 AutoChooser::entry_t Struct Reference	42
5.18.1 Detailed Description	42

5.18.2 Member Data Documentation . . . . .	43
5.18.2.1 name . . . . .	43
5.19 ExponentialMovingAverage Class Reference . . . . .	43
5.19.1 Detailed Description . . . . .	43
5.19.2 Constructor & Destructor Documentation . . . . .	43
5.19.2.1 ExponentialMovingAverage() [1/2] . . . . .	43
5.19.2.2 ExponentialMovingAverage() [2/2] . . . . .	44
5.19.3 Member Function Documentation . . . . .	44
5.19.3.1 add_entry() . . . . .	44
5.19.3.2 get_size() . . . . .	44
5.19.3.3 get_value() . . . . .	45
5.20 Feedback Class Reference . . . . .	45
5.20.1 Detailed Description . . . . .	45
5.20.2 Member Function Documentation . . . . .	46
5.20.2.1 get() . . . . .	46
5.20.2.2 init() . . . . .	46
5.20.2.3 is_on_target() . . . . .	46
5.20.2.4 set_limits() . . . . .	46
5.20.2.5 update() . . . . .	47
5.21 FeedForward Class Reference . . . . .	47
5.21.1 Detailed Description . . . . .	48
5.21.2 Constructor & Destructor Documentation . . . . .	48
5.21.2.1 FeedForward() . . . . .	48
5.21.3 Member Function Documentation . . . . .	48
5.21.3.1 calculate() . . . . .	48
5.22 FeedForward::ff_config_t Struct Reference . . . . .	49
5.22.1 Detailed Description . . . . .	49
5.22.2 Member Data Documentation . . . . .	49
5.22.2.1 kA . . . . .	49
5.22.2.2 kG . . . . .	50
5.22.2.3 kS . . . . .	50
5.22.2.4 kV . . . . .	50
5.23 Filter Class Reference . . . . .	50
5.23.1 Detailed Description . . . . .	50
5.23.2 Member Function Documentation . . . . .	51
5.23.2.1 add_entry() . . . . .	51
5.23.2.2 get_value() . . . . .	51
5.24 Flywheel Class Reference . . . . .	51
5.24.1 Detailed Description . . . . .	52
5.24.2 Constructor & Destructor Documentation . . . . .	52
5.24.2.1 Flywheel() . . . . .	52
5.24.3 Member Function Documentation . . . . .	52

---

5.24.3.1 get_motors() . . . . .	52
5.24.3.2 get_target() . . . . .	52
5.24.3.3 getRPM() . . . . .	53
5.24.3.4 is_on_target() . . . . .	53
5.24.3.5 Page() . . . . .	53
5.24.3.6 spin_manual() . . . . .	53
5.24.3.7 spin_rpm() . . . . .	54
5.24.3.8 SpinRpmCmd() . . . . .	54
5.24.3.9 stop() . . . . .	54
5.24.3.10 WaitUntilUpToSpeedCmd() . . . . .	54
5.24.4 Friends And Related Symbol Documentation . . . . .	55
5.24.4.1 spinRPMTask . . . . .	55
5.25 FlywheelPage Class Reference . . . . .	55
5.25.1 Member Function Documentation . . . . .	55
5.25.1.1 draw() . . . . .	55
5.25.1.2 update() . . . . .	56
5.26 FlywheelStopCommand Class Reference . . . . .	56
5.26.1 Detailed Description . . . . .	57
5.26.2 Constructor & Destructor Documentation . . . . .	57
5.26.2.1 FlywheelStopCommand() . . . . .	57
5.26.3 Member Function Documentation . . . . .	57
5.26.3.1 run() . . . . .	57
5.27 FlywheelStopMotorsCommand Class Reference . . . . .	57
5.27.1 Detailed Description . . . . .	58
5.27.2 Constructor & Destructor Documentation . . . . .	58
5.27.2.1 FlywheelStopMotorsCommand() . . . . .	58
5.27.3 Member Function Documentation . . . . .	58
5.27.3.1 run() . . . . .	58
5.28 FlywheelStopNonTasksCommand Class Reference . . . . .	59
5.28.1 Detailed Description . . . . .	59
5.29 FunctionCommand Class Reference . . . . .	60
5.29.1 Detailed Description . . . . .	60
5.29.2 Member Function Documentation . . . . .	61
5.29.2.1 run() . . . . .	61
5.30 FunctionCondition Class Reference . . . . .	61
5.30.1 Detailed Description . . . . .	61
5.30.2 Member Function Documentation . . . . .	62
5.30.2.1 test() . . . . .	62
5.31 screen::FunctionPage Class Reference . . . . .	62
5.31.1 Detailed Description . . . . .	62
5.31.2 Constructor & Destructor Documentation . . . . .	62
5.31.2.1 FunctionPage() . . . . .	62

5.31.3 Member Function Documentation . . . . .	63
5.31.3.1 draw() . . . . .	63
5.31.3.2 update() . . . . .	63
5.32 GenericAuto Class Reference . . . . .	63
5.32.1 Detailed Description . . . . .	64
5.32.2 Member Function Documentation . . . . .	64
5.32.2.1 add() . . . . .	64
5.32.2.2 add_async() . . . . .	64
5.32.2.3 add_delay() . . . . .	64
5.32.2.4 run() . . . . .	65
5.33 GraphDrawer Class Reference . . . . .	65
5.33.1 Constructor & Destructor Documentation . . . . .	65
5.33.1.1 GraphDrawer() . . . . .	65
5.33.2 Member Function Documentation . . . . .	66
5.33.2.1 add_samples() [1/2] . . . . .	66
5.33.2.2 add_samples() [2/2] . . . . .	66
5.33.2.3 draw() . . . . .	66
5.34 PurePursuit::hermite_point Struct Reference . . . . .	67
5.34.1 Detailed Description . . . . .	67
5.35 IfTimePassed Class Reference . . . . .	67
5.35.1 Detailed Description . . . . .	68
5.35.2 Member Function Documentation . . . . .	68
5.35.2.1 test() . . . . .	68
5.36 InOrder Class Reference . . . . .	68
5.36.1 Detailed Description . . . . .	69
5.36.2 Member Function Documentation . . . . .	69
5.36.2.1 on_timeout() . . . . .	69
5.36.2.2 run() . . . . .	69
5.37 Lift< T > Class Template Reference . . . . .	70
5.37.1 Detailed Description . . . . .	70
5.37.2 Constructor & Destructor Documentation . . . . .	70
5.37.2.1 Lift() . . . . .	70
5.37.3 Member Function Documentation . . . . .	71
5.37.3.1 control_continuous() . . . . .	71
5.37.3.2 control_manual() . . . . .	71
5.37.3.3 control_setpoints() . . . . .	71
5.37.3.4 get_async() . . . . .	72
5.37.3.5 get_setpoint() . . . . .	72
5.37.3.6 hold() . . . . .	72
5.37.3.7 home() . . . . .	72
5.37.3.8 set_async() . . . . .	72
5.37.3.9 set_position() . . . . .	73

---

5.37.3.10 set_sensor_function() . . . . .	73
5.37.3.11 set_sensor_reset() . . . . .	73
5.37.3.12 set_setpoint() . . . . .	73
5.38 Lift< T >::lift_cfg_t Struct Reference . . . . .	74
5.38.1 Detailed Description . . . . .	74
5.39 Logger Class Reference . . . . .	74
5.39.1 Detailed Description . . . . .	75
5.39.2 Constructor & Destructor Documentation . . . . .	75
5.39.2.1 Logger() . . . . .	75
5.39.3 Member Function Documentation . . . . .	76
5.39.3.1 Log() [1/2] . . . . .	76
5.39.3.2 Log() [2/2] . . . . .	76
5.39.3.3 Logf() [1/2] . . . . .	76
5.39.3.4 Logf() [2/2] . . . . .	76
5.39.3.5 LogIn() [1/2] . . . . .	77
5.39.3.6 LogIn() [2/2] . . . . .	77
5.40 MotionController::m_profile_cfg_t Struct Reference . . . . .	77
5.40.1 Detailed Description . . . . .	78
5.41 Mat2 Struct Reference . . . . .	78
5.42 MecanumDrive Class Reference . . . . .	78
5.42.1 Detailed Description . . . . .	79
5.42.2 Constructor & Destructor Documentation . . . . .	79
5.42.2.1 MecanumDrive() . . . . .	79
5.42.3 Member Function Documentation . . . . .	79
5.42.3.1 auto_drive() . . . . .	79
5.42.3.2 auto_turn() . . . . .	80
5.42.3.3 drive() . . . . .	80
5.42.3.4 drive_raw() . . . . .	81
5.43 MecanumDrive::mecanumdrive_config_t Struct Reference . . . . .	82
5.43.1 Detailed Description . . . . .	82
5.44 motion_t Struct Reference . . . . .	82
5.44.1 Detailed Description . . . . .	82
5.45 MotionController Class Reference . . . . .	83
5.45.1 Detailed Description . . . . .	83
5.45.2 Constructor & Destructor Documentation . . . . .	84
5.45.2.1 MotionController() . . . . .	84
5.45.3 Member Function Documentation . . . . .	84
5.45.3.1 get() . . . . .	84
5.45.3.2 get_motion() . . . . .	84
5.45.3.3 init() . . . . .	84
5.45.3.4 is_on_target() . . . . .	85
5.45.3.5 set_limits() . . . . .	85

5.45.3.6 tune_feedforward()	85
5.45.3.7 update()	86
5.46 MovingAverage Class Reference	86
5.46.1 Detailed Description	87
5.46.2 Constructor & Destructor Documentation	87
5.46.2.1 MovingAverage() [1/2]	87
5.46.2.2 MovingAverage() [2/2]	87
5.46.3 Member Function Documentation	88
5.46.3.1 add_entry()	88
5.46.3.2 get_size()	88
5.46.3.3 get_value()	88
5.47 Odometry3Wheel Class Reference	89
5.47.1 Detailed Description	90
5.47.2 Constructor & Destructor Documentation	90
5.47.2.1 Odometry3Wheel()	90
5.47.3 Member Function Documentation	91
5.47.3.1 tune()	91
5.47.3.2 update()	91
5.48 Odometry3Wheel::odometry3wheel_cfg_t Struct Reference	92
5.48.1 Detailed Description	92
5.48.2 Member Data Documentation	92
5.48.2.1 off_axis_center_dist	92
5.48.2.2 wheel_diam	92
5.48.2.3 wheelbase_dist	92
5.49 OdometryBase Class Reference	92
5.49.1 Detailed Description	93
5.49.2 Constructor & Destructor Documentation	94
5.49.2.1 OdometryBase()	94
5.49.3 Member Function Documentation	94
5.49.3.1 background_task()	94
5.49.3.2 end_async()	94
5.49.3.3 get_accel()	94
5.49.3.4 get_angular_accel_deg()	95
5.49.3.5 get_angular_speed_deg()	95
5.49.3.6 get_position()	95
5.49.3.7 get_speed()	95
5.49.3.8 pos_diff()	95
5.49.3.9 rot_diff()	96
5.49.3.10 set_position()	96
5.49.3.11 smallest_angle()	96
5.49.3.12 update()	97
5.49.4 Member Data Documentation	97

---

5.49.4.1 accel . . . . .	97
5.49.4.2 ang_accel_deg . . . . .	97
5.49.4.3 ang_speed_deg . . . . .	97
5.49.4.4 current_pos . . . . .	98
5.49.4.5 handle . . . . .	98
5.49.4.6 mut . . . . .	98
5.49.4.7 speed . . . . .	98
5.49.4.8 zero_pos . . . . .	98
5.50 screen::OdometryPage Class Reference . . . . .	98
5.50.1 Detailed Description . . . . .	99
5.50.2 Constructor & Destructor Documentation . . . . .	99
5.50.2.1 OdometryPage() . . . . .	99
5.50.3 Member Function Documentation . . . . .	99
5.50.3.1 draw() . . . . .	99
5.50.3.2 update() . . . . .	100
5.51 OdometryTank Class Reference . . . . .	100
5.51.1 Detailed Description . . . . .	101
5.51.2 Constructor & Destructor Documentation . . . . .	101
5.51.2.1 OdometryTank() [1/3] . . . . .	101
5.51.2.2 OdometryTank() [2/3] . . . . .	102
5.51.2.3 OdometryTank() [3/3] . . . . .	102
5.51.3 Member Function Documentation . . . . .	103
5.51.3.1 set_position() . . . . .	103
5.51.3.2 update() . . . . .	103
5.52 OdomSetPosition Class Reference . . . . .	103
5.52.1 Detailed Description . . . . .	104
5.52.2 Constructor & Destructor Documentation . . . . .	104
5.52.2.1 OdomSetPosition() . . . . .	104
5.52.3 Member Function Documentation . . . . .	105
5.52.3.1 run() . . . . .	105
5.53 OrCondition Class Reference . . . . .	105
5.53.1 Member Function Documentation . . . . .	105
5.53.1.1 test() . . . . .	105
5.54 screen::Page Class Reference . . . . .	106
5.54.1 Detailed Description . . . . .	106
5.54.2 Member Function Documentation . . . . .	106
5.54.2.1 draw() . . . . .	106
5.54.2.2 update() . . . . .	106
5.55 Parallel Class Reference . . . . .	107
5.55.1 Detailed Description . . . . .	108
5.55.2 Member Function Documentation . . . . .	108
5.55.2.1 on_timeout() . . . . .	108

---

5.55.2.2 run()	108
5.56 parallel_runner_info Struct Reference	108
5.57 PurePursuit::Path Class Reference	109
5.57.1 Detailed Description	109
5.57.2 Constructor & Destructor Documentation	109
5.57.2.1 Path()	109
5.57.3 Member Function Documentation	109
5.57.3.1 get_points()	109
5.57.3.2 get_radius()	109
5.57.3.3 is_valid()	110
5.58 PID Class Reference	110
5.58.1 Detailed Description	111
5.58.2 Member Enumeration Documentation	111
5.58.2.1 ERROR_TYPE	111
5.58.3 Constructor & Destructor Documentation	111
5.58.3.1 PID()	111
5.58.4 Member Function Documentation	112
5.58.4.1 get()	112
5.58.4.2 get_error()	112
5.58.4.3 get_sensor_val()	112
5.58.4.4 get_target()	112
5.58.4.5 init()	112
5.58.4.6 is_on_target()	113
5.58.4.7 reset()	113
5.58.4.8 set_limits()	113
5.58.4.9 set_target()	114
5.58.4.10 update()	114
5.59 PID::pid_config_t Struct Reference	114
5.59.1 Detailed Description	115
5.60 PIDFF Class Reference	115
5.60.1 Member Function Documentation	116
5.60.1.1 get()	116
5.60.1.2 init()	116
5.60.1.3 is_on_target()	116
5.60.1.4 set_limits()	117
5.60.1.5 set_target()	117
5.60.1.6 update() [1/2]	117
5.60.1.7 update() [2/2]	117
5.61 screen::PIDPage Class Reference	118
5.61.1 Detailed Description	118
5.61.2 Constructor & Destructor Documentation	118
5.61.2.1 PIDPage()	118

---

5.61.3 Member Function Documentation . . . . .	119
5.61.3.1 draw() . . . . .	119
5.61.3.2 update() . . . . .	119
5.62 point_t Struct Reference . . . . .	119
5.62.1 Detailed Description . . . . .	120
5.62.2 Member Function Documentation . . . . .	120
5.62.2.1 dist() . . . . .	120
5.62.2.2 operator+() . . . . .	120
5.62.2.3 operator-() . . . . .	121
5.63 pose_t Struct Reference . . . . .	121
5.63.1 Detailed Description . . . . .	122
5.64 PurePursuitCommand Class Reference . . . . .	122
5.64.1 Detailed Description . . . . .	122
5.64.2 Constructor & Destructor Documentation . . . . .	123
5.64.2.1 PurePursuitCommand() . . . . .	123
5.64.3 Member Function Documentation . . . . .	123
5.64.3.1 on_timeout() . . . . .	123
5.64.3.2 run() . . . . .	123
5.65 Rect Struct Reference . . . . .	123
5.66 RepeatUntil Class Reference . . . . .	124
5.66.1 Constructor & Destructor Documentation . . . . .	125
5.66.1.1 RepeatUntil() [1/2] . . . . .	125
5.66.1.2 RepeatUntil() [2/2] . . . . .	126
5.66.2 Member Function Documentation . . . . .	126
5.66.2.1 on_timeout() . . . . .	126
5.66.2.2 run() . . . . .	126
5.67 robot_specs_t Struct Reference . . . . .	127
5.67.1 Detailed Description . . . . .	127
5.68 screen::ScreenData Struct Reference . . . . .	127
5.68.1 Detailed Description . . . . .	128
5.69 Serializer Class Reference . . . . .	128
5.69.1 Detailed Description . . . . .	128
5.69.2 Constructor & Destructor Documentation . . . . .	128
5.69.2.1 Serializer() . . . . .	128
5.69.3 Member Function Documentation . . . . .	129
5.69.3.1 bool_or() . . . . .	129
5.69.3.2 double_or() . . . . .	129
5.69.3.3 int_or() . . . . .	129
5.69.3.4 save_to_disk() . . . . .	131
5.69.3.5 set_bool() . . . . .	131
5.69.3.6 set_double() . . . . .	131
5.69.3.7 set_int() . . . . .	131

5.69.3.8 set_string() . . . . .	132
5.69.3.9 string_or() . . . . .	132
5.70 screen::SliderWidget Class Reference . . . . .	133
5.70.1 Detailed Description . . . . .	133
5.70.2 Constructor & Destructor Documentation . . . . .	133
5.70.2.1 SliderWidget() . . . . .	133
5.70.3 Member Function Documentation . . . . .	133
5.70.3.1 update() . . . . .	133
5.71 SpinRPMCommand Class Reference . . . . .	134
5.71.1 Detailed Description . . . . .	135
5.71.2 Constructor & Destructor Documentation . . . . .	135
5.71.2.1 SpinRPMCommand() . . . . .	135
5.71.3 Member Function Documentation . . . . .	135
5.71.3.1 run() . . . . .	135
5.72 PurePursuit::spline Struct Reference . . . . .	136
5.72.1 Detailed Description . . . . .	136
5.73 screen::StatsPage Class Reference . . . . .	136
5.73.1 Detailed Description . . . . .	137
5.73.2 Constructor & Destructor Documentation . . . . .	137
5.73.2.1 StatsPage() . . . . .	137
5.73.3 Member Function Documentation . . . . .	137
5.73.3.1 draw() . . . . .	137
5.73.3.2 update() . . . . .	137
5.74 TakeBackHalf Class Reference . . . . .	138
5.74.1 Detailed Description . . . . .	138
5.74.2 Member Function Documentation . . . . .	138
5.74.2.1 get() . . . . .	138
5.74.2.2 init() . . . . .	138
5.74.2.3 is_on_target() . . . . .	139
5.74.2.4 set_limits() . . . . .	139
5.74.2.5 update() . . . . .	139
5.75 TankDrive Class Reference . . . . .	140
5.75.1 Detailed Description . . . . .	141
5.75.2 Constructor & Destructor Documentation . . . . .	141
5.75.2.1 TankDrive() . . . . .	141
5.75.3 Member Function Documentation . . . . .	141
5.75.3.1 drive_arcade() . . . . .	141
5.75.3.2 drive_forward() [1/2] . . . . .	142
5.75.3.3 drive_forward() [2/2] . . . . .	142
5.75.3.4 drive_tank() . . . . .	143
5.75.3.5 drive_to_point() [1/2] . . . . .	144
5.75.3.6 drive_to_point() [2/2] . . . . .	144

---

5.75.3.7 modify_inputs() . . . . .	145
5.75.3.8 pure_pursuit() [1/2] . . . . .	146
5.75.3.9 pure_pursuit() [2/2] . . . . .	146
5.75.3.10 reset_auto() . . . . .	147
5.75.3.11 stop() . . . . .	147
5.75.3.12 turn_degrees() [1/2] . . . . .	148
5.75.3.13 turn_degrees() [2/2] . . . . .	149
5.75.3.14 turn_to_heading() [1/2] . . . . .	150
5.75.3.15 turn_to_heading() [2/2] . . . . .	150
5.76 TimesTestedCondition Class Reference . . . . .	151
5.76.1 Member Function Documentation . . . . .	151
5.76.1.1 test() . . . . .	151
5.77 trapezoid_profile_segment_t Struct Reference . . . . .	152
5.77.1 Detailed Description . . . . .	152
5.78 TrapezoidProfile Class Reference . . . . .	152
5.78.1 Detailed Description . . . . .	153
5.78.2 Constructor & Destructor Documentation . . . . .	153
5.78.2.1 TrapezoidProfile() . . . . .	153
5.78.3 Member Function Documentation . . . . .	154
5.78.3.1 calculate() . . . . .	154
5.78.3.2 calculate_time_based() . . . . .	154
5.78.3.3 get_movement_time() . . . . .	154
5.78.3.4 set_accel() . . . . .	155
5.78.3.5 set_endpts() . . . . .	155
5.78.3.6 set_max_v() . . . . .	155
5.78.3.7 set_vel_endpts() . . . . .	155
5.79 TurnDegreesCommand Class Reference . . . . .	156
5.79.1 Detailed Description . . . . .	157
5.79.2 Constructor & Destructor Documentation . . . . .	157
5.79.2.1 TurnDegreesCommand() . . . . .	157
5.79.3 Member Function Documentation . . . . .	157
5.79.3.1 on_timeout() . . . . .	157
5.79.3.2 run() . . . . .	157
5.80 TurnToHeadingCommand Class Reference . . . . .	158
5.80.1 Detailed Description . . . . .	158
5.80.2 Constructor & Destructor Documentation . . . . .	158
5.80.2.1 TurnToHeadingCommand() . . . . .	158
5.80.3 Member Function Documentation . . . . .	159
5.80.3.1 on_timeout() . . . . .	159
5.80.3.2 run() . . . . .	159
5.81 Vector2D Class Reference . . . . .	159
5.81.1 Detailed Description . . . . .	160

5.81.2 Constructor & Destructor Documentation . . . . .	160
5.81.2.1 Vector2D() [1/2] . . . . .	160
5.81.2.2 Vector2D() [2/2] . . . . .	160
5.81.3 Member Function Documentation . . . . .	161
5.81.3.1 get_dir() . . . . .	161
5.81.3.2 get_mag() . . . . .	161
5.81.3.3 get_x() . . . . .	161
5.81.3.4 get_y() . . . . .	161
5.81.3.5 normalize() . . . . .	162
5.81.3.6 operator*() . . . . .	162
5.81.3.7 operator+() . . . . .	162
5.81.3.8 operator-() . . . . .	162
5.81.3.9 point() . . . . .	163
5.82 WaitUntilCondition Class Reference . . . . .	163
5.82.1 Detailed Description . . . . .	164
5.82.2 Member Function Documentation . . . . .	164
5.82.2.1 run() . . . . .	164
5.83 WaitUntilUpToSpeedCommand Class Reference . . . . .	164
5.83.1 Detailed Description . . . . .	165
5.83.2 Constructor & Destructor Documentation . . . . .	165
5.83.2.1 WaitUntilUpToSpeedCommand() . . . . .	165
5.83.3 Member Function Documentation . . . . .	166
5.83.3.1 run() . . . . .	166
<b>6 File Documentation</b> . . . . .	<b>167</b>
6.1 robot_specs.h . . . . .	167
6.2 custom_encoder.h . . . . .	167
6.3 flywheel.h . . . . .	168
6.4 lift.h . . . . .	168
6.5 mecanum_drive.h . . . . .	171
6.6 odometry_3wheel.h . . . . .	172
6.7 odometry_base.h . . . . .	172
6.8 odometry_tank.h . . . . .	173
6.9 screen.h . . . . .	174
6.10 tank_drive.h . . . . .	176
6.11 auto_chooser.h . . . . .	177
6.12 auto_command.h . . . . .	177
6.13 basic_command.h . . . . .	179
6.14 command_controller.h . . . . .	180
6.15 delay_command.h . . . . .	181
6.16 drive_commands.h . . . . .	181
6.17 flywheel_commands.h . . . . .	183

---

6.18 bang_bang.h . . . . .	184
6.19 feedback_base.h . . . . .	184
6.20 feedforward.h . . . . .	184
6.21 motion_controller.h . . . . .	185
6.22 pid.h . . . . .	185
6.23 pidff.h . . . . .	186
6.24 take_back_half.h . . . . .	187
6.25 trapezoid_profile.h . . . . .	187
6.26 generic_auto.h . . . . .	188
6.27 geometry.h . . . . .	189
6.28 graph_drawer.h . . . . .	190
6.29 logger.h . . . . .	190
6.30 math_util.h . . . . .	191
6.31 moving_average.h . . . . .	192
6.32 pure_pursuit.h . . . . .	193
6.33 serializer.h . . . . .	193
6.34 vector2d.h . . . . .	194
<b>Index</b>	<b>197</b>



# Chapter 1

## Core

This is the host repository for the custom VEX libraries used by the RIT VEXU team

Automatically updated documentation is available at [here](#). There is also a downloadable [reference manual](#).

### 1.1 Getting Started

In order to simply use this repo, you can either clone it into your VEXcode project folder, or download the .zip and place it into a core/ subfolder. Then follow the instructions for setting up compilation at [Wiki/BuildSystem](#)

If you wish to contribute, follow the instructions at [Wiki/ProjectSetup](#)

### 1.2 Features

Here is the current feature list this repo provides:

Subsystems (See [Wiki/Subsystems](#)):

- Tank drivetrain (user control / autonomous)
- Mecanum drivetrain (user control / autonomous)
- Odometry
- [Flywheel](#)
- [Lift](#)
- Custom encoders

Utilities (See [Wiki/Utilites](#)):

- [PID](#) controller
- [FeedForward](#) controller
- Trapezoidal motion profile controller
- Pure Pursuit
- Generic auto program builder
- Auto program UI selector
- Mathematical classes ([Vector2D](#), Moving Average)



# Chapter 2

## Hierarchical Index

### 2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

AutoCommand . . . . .	16
Async . . . . .	12
BasicSolenoidSet . . . . .	20
BasicSpinCommand . . . . .	21
BasicStopCommand . . . . .	23
Branch . . . . .	25
DelayCommand . . . . .	35
DriveForwardCommand . . . . .	36
DriveStopCommand . . . . .	38
DriveToPointCommand . . . . .	40
FlywheelStopCommand . . . . .	56
FlywheelStopMotorsCommand . . . . .	57
FlywheelStopNonTasksCommand . . . . .	59
FunctionCommand . . . . .	60
InOrder . . . . .	68
OdomSetPosition . . . . .	103
Parallel . . . . .	107
PurePursuitCommand . . . . .	122
RepeatUntil . . . . .	124
SpinRPMCommand . . . . .	134
TurnDegreesCommand . . . . .	156
TurnToHeadingCommand . . . . .	158
WaitUntilCondition . . . . .	163
WaitUntilUpToSpeedCommand . . . . .	164
screen::ButtonWidget . . . . .	26
CommandController . . . . .	28
Condition . . . . .	32
AndCondition . . . . .	11
FunctionCondition . . . . .	61
IfTimePassed . . . . .	67
OrCondition . . . . .	105
TimesTestedCondition . . . . .	151
vex::encoder	
CustomEncoder . . . . .	32
AutoChooser::entry_t . . . . .	42

Feedback . . . . .	45
BangBang . . . . .	18
MotionController . . . . .	83
PID . . . . .	110
PIDFF . . . . .	115
TakeBackHalf . . . . .	138
FeedForward . . . . .	47
FeedForward::ff_config_t . . . . .	49
Filter . . . . .	50
ExponentialMovingAverage . . . . .	43
MovingAverage . . . . .	86
Flywheel . . . . .	51
GenericAuto . . . . .	63
GraphDrawer . . . . .	65
PurePursuit::hermite_point . . . . .	67
Lift< T > . . . . .	70
Lift< T >::lift_cfg_t . . . . .	74
Logger . . . . .	74
MotionController::m_profile_cfg_t . . . . .	77
Mat2 . . . . .	78
MecanumDrive . . . . .	78
MecanumDrive::mecanumdrive_config_t . . . . .	82
motion_t . . . . .	82
Odometry3Wheel::odometry3wheel_cfg_t . . . . .	92
OdometryBase . . . . .	92
Odometry3Wheel . . . . .	89
OdometryTank . . . . .	100
screen::Page . . . . .	106
AutoChooser . . . . .	13
FlywheelPage . . . . .	55
screen::FunctionPage . . . . .	62
screen::OdometryPage . . . . .	98
screen::PIDPage . . . . .	118
screen::StatsPage . . . . .	136
parallel_runner_info . . . . .	108
PurePursuit::Path . . . . .	109
PID::pid_config_t . . . . .	114
point_t . . . . .	119
pose_t . . . . .	121
Rect . . . . .	123
robot_specs_t . . . . .	127
screen::ScreenData . . . . .	127
Serializer . . . . .	128
screen::SliderWidget . . . . .	133
PurePursuit::spline . . . . .	136
TankDrive . . . . .	140
trapezoid_profile_segment_t . . . . .	152
TrapezoidProfile . . . . .	152
Vector2D . . . . .	159

# Chapter 3

## Class Index

### 3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

AndCondition . . . . .	11
Async	
<b>Async</b> runs a command asynchronously will simply let it go and never look back THIS HAS A VERY NICHE USE CASE. THINK ABOUT IF YOU REALLY NEED IT . . . . .	12
AutoChooser . . . . .	13
AutoCommand . . . . .	16
BangBang . . . . .	18
BasicSolenoidSet . . . . .	20
BasicSpinCommand . . . . .	21
BasicStopCommand . . . . .	23
Branch	
<b>Branch</b> chooses from multiple options at runtime. the function decider returns an index into the choices vector If you wish to make no choice and skip this section, return NO_CHOICE; any choice that is out of bounds set to NO_CHOICE . . . . .	25
screen::ButtonWidget	
Widget that does something when you tap it. The function is only called once when you first tap it	26
CommandController . . . . .	28
Condition . . . . .	32
CustomEncoder . . . . .	32
DelayCommand . . . . .	35
DriveForwardCommand . . . . .	36
DriveStopCommand . . . . .	38
DriveToPointCommand . . . . .	40
AutoChooser::entry_t . . . . .	42
ExponentialMovingAverage . . . . .	43
Feedback . . . . .	45
FeedForward . . . . .	47
FeedForward::ff_config_t . . . . .	49
Filter . . . . .	50
Flywheel . . . . .	51
FlywheelPage . . . . .	55
FlywheelStopCommand . . . . .	56
FlywheelStopMotorsCommand . . . . .	57
FlywheelStopNonTasksCommand . . . . .	59
FunctionCommand . . . . .	60

FunctionCondition	
FunctionCondition	is a quick and dirty <a href="#">Condition</a> to wrap some expression that should be evaluated at runtime . . . . .
screen::FunctionPage	61
Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything . . . . .	62
GenericAuto	63
GraphDrawer	65
PurePursuit::hermite_point	67
IfTimePassed	
IfTimePassed	tests based on time since the command controller was constructed. Returns true if elapsed time > time_s . . . . .
InOrder	67
InOrder	runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timeouts? . . . . .
Lift< T >	68
Lift< T >::lift_cfg_t	70
Logger	74
Class to simplify writing to files	74
MotionController::m_profile_cfg_t	77
Mat2	78
MecanumDrive	78
MecanumDrive::mecanumdrive_config_t	82
motion_t	82
MotionController	83
MovingAverage	86
Odometry3Wheel	89
Odometry3Wheel::odometry3wheel_cfg_t	92
OdometryBase	92
screen::OdometryPage	
Page	that shows odometry position and rotation and a map (if an sd card with the file is on) . . . . .
OdometryTank	98
OdomSetPosition	100
OrCondition	103
screen::Page	
Page	describes one part of the screen slideshow . . . . .
Parallel	106
Parallel	runs multiple commands in parallel and waits for all to finish before continuing. if none finish before this command's timeout, it will call on_timeout on all children continue . . . . .
parallel_runner_info	107
PurePursuit::Path	108
PID	109
PID::pid_config_t	110
PIDFF	114
screen::PIDPage	
PIDPage	provides a way to tune a pid controller on the screen . . . . .
point_t	118
pose_t	119
PurePursuitCommand	121
Rect	122
RepeatUntil	123
robot_specs_t	124
screen::ScreenData	
Holds the data that will be passed to the screen thread you probably shouldnt have to use it	127
Serializer	
Serializes Arbitrary data to a file on the SD Card . . . . .	128

screen::SliderWidget	Widget that updates a double value. Updates by reference so watch out for race conditions cuz the screen stuff lives on another thread . . . . .	133
SpinRPMCommand	. . . . .	134
PurePursuit::spline	. . . . .	136
screen::StatsPage	Draws motor stats and battery stats to the screen . . . . .	136
TakeBackHalf	A velocity controller . . . . .	138
TankDrive	. . . . .	140
TimesTestedCondition	. . . . .	151
trapezoid_profile_segment_t	. . . . .	152
TrapezoidProfile	. . . . .	152
TurnDegreesCommand	. . . . .	156
TurnToHeadingCommand	. . . . .	158
Vector2D	. . . . .	159
WaitUntilCondition	Waits until the condition is true . . . . .	163
WaitUntilUpToSpeedCommand	. . . . .	164



# Chapter 4

## File Index

### 4.1 File List

Here is a list of all documented files with brief descriptions:

include/robot_specs.h . . . . .	167
include/subsystems/custom_encoder.h . . . . .	167
include/subsystems/flywheel.h . . . . .	168
include/subsystems/lift.h . . . . .	168
include/subsystems/mecanum_drive.h . . . . .	171
include/subsystems/screen.h . . . . .	174
include/subsystems/tank_drive.h . . . . .	176
include/subsystems/odometry/odometry_3wheel.h . . . . .	172
include/subsystems/odometry/odometry_base.h . . . . .	172
include/subsystems/odometry/odometry_tank.h . . . . .	173
include/utils/auto_chooser.h . . . . .	177
include/utils/generic_auto.h . . . . .	188
include/utils/geometry.h . . . . .	189
include/utils/graph_drawer.h . . . . .	190
include/utils/logger.h . . . . .	190
include/utils/math_util.h . . . . .	191
include/utils/moving_average.h . . . . .	192
include/utils/pure_pursuit.h . . . . .	193
include/utils/serializer.h . . . . .	193
include/utils/vector2d.h . . . . .	194
include/utils/command_structure/auto_command.h . . . . .	177
include/utils/command_structure/basic_command.h . . . . .	179
include/utils/command_structure/command_controller.h . . . . .	180
include/utils/command_structure/delay_command.h . . . . .	181
include/utils/command_structure/drive_commands.h . . . . .	181
include/utils/command_structure/flywheel_commands.h . . . . .	183
include/utils/controls/bang_bang.h . . . . .	184
include/utils/controls/feedback_base.h . . . . .	184
include/utils/controls/feedforward.h . . . . .	184
include/utils/controls/motion_controller.h . . . . .	185
include/utils/controls/pid.h . . . . .	185
include/utils/controls/pidff.h . . . . .	186
include/utils/controls/take_back_half.h . . . . .	187
include/utils/controls/trapezoid_profile.h . . . . .	187

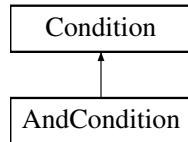


# Chapter 5

## Class Documentation

### 5.1 AndCondition Class Reference

Inheritance diagram for AndCondition:



#### Public Member Functions

- **AndCondition** ([Condition](#) \*A, [Condition](#) \*B)
- bool [test](#) () override

#### Public Member Functions inherited from [Condition](#)

- [Condition](#) \* **Or** ([Condition](#) \*b)
- [Condition](#) \* **And** ([Condition](#) \*b)

#### 5.1.1 Member Function Documentation

##### 5.1.1.1 [test\(\)](#)

```
bool AndCondition::test ( ) [inline], [override], [virtual]
```

Implements [Condition](#).

The documentation for this class was generated from the following file:

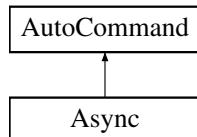
- src/utils/command\_structure/auto\_command.cpp

## 5.2 Async Class Reference

[Async](#) runs a command asynchronously will simply let it go and never look back THIS HAS A VERY NICHE USE CASE. THINK ABOUT IF YOU REALLY NEED IT.

```
#include <auto_command.h>
```

Inheritance diagram for Async:



### Public Member Functions

- **Async** ([AutoCommand](#) \*cmd)
- bool [run](#) () override

### Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

### Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

#### 5.2.1 Detailed Description

[Async](#) runs a command asynchronously will simply let it go and never look back THIS HAS A VERY NICHE USE CASE. THINK ABOUT IF YOU REALLY NEED IT.

## 5.2.2 Member Function Documentation

### 5.2.2.1 run()

```
bool Async::run ( ) [override], [virtual]
```

Executes the command Overridden by child classes

Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

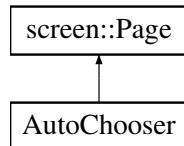
The documentation for this class was generated from the following files:

- [include/utils/command\\_structure/auto\\_command.h](#)
- [src/utils/command\\_structure/auto\\_command.cpp](#)

## 5.3 AutoChooser Class Reference

```
#include <auto_chooser.h>
```

Inheritance diagram for AutoChooser:



### Classes

- struct [entry\\_t](#)

### Public Member Functions

- [AutoChooser](#) (`std::vector< std::string > paths, size_t def=0)`
- void [update](#) (`bool was_pressed, int x, int y)`  
*collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this Page (only drawn page gets touch updates))*
- void [draw](#) (`vex::brain::lcd &, bool first_draw, unsigned int frame_number)`  
*draw stored data to the screen (runs at 10 hz and only runs if this page is in front)*
- `size_t get\_choice ()`

### Protected Attributes

- `size_t choice`
- `std::vector< entry\_t > list`

## Static Protected Attributes

- static const size\_t **width** = 380
- static const size\_t **height** = 220

### 5.3.1 Detailed Description

Autochooser is a utility to make selecting robot autonomous programs easier source: RIT VexU Wiki During a season, we usually code between 4 and 6 autonomous programs. Most teams will change their entire robot program as a way of choosing autonomy but this may cause issues if you have an emergency patch to upload during a competition. This class was built as a way of using the robot screen to list autonomous programs, and the touchscreen to select them.

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 AutoChooser()

```
AutoChooser::AutoChooser (
    std::vector< std::string > paths,
    size_t def = 0 )
```

Initialize the auto-chooser. This class places a choice menu on the brain screen, so the driver can choose which autonomous to run.

#### Parameters

<i>brain</i>	the brain on which to draw the selection boxes
--------------	--

### 5.3.3 Member Function Documentation

#### 5.3.3.1 draw()

```
void AutoChooser::draw (
    vex::brain::lcd & screen,
    bool first_draw,
    unsigned int frame_number ) [virtual]
```

draw stored data to the screen (runs at 10 hz and only runs if this page is in front)

#### Parameters

<i>first_draw</i>	true if we just switched to this page
<i>frame_number</i>	frame of drawing we are on (basically an animation tick)

Reimplemented from [screen::Page](#).

### 5.3.3.2 get\_choice()

```
size_t AutoChooser::get_choice ( )
```

Get the currently selected auto choice

#### Returns

the identifier to the auto path

Return the selected autonomous

### 5.3.3.3 update()

```
void AutoChooser::update (
    bool was_pressed,
    int x,
    int y ) [virtual]
```

collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this Page (only drawn page gets touch updates))

#### Parameters

<i>was_pressed</i>	true if the screen has been pressed
<i>x</i>	x position of screen press (if the screen was pressed)
<i>y</i>	y position of screen press (if the screen was pressed)

Reimplemented from [screen::Page](#).

## 5.3.4 Member Data Documentation

### 5.3.4.1 choice

```
size_t AutoChooser::choice [protected]
```

the current choice of auto

### 5.3.4.2 list

```
std::vector<entry\_t> AutoChooser::list [protected]
```

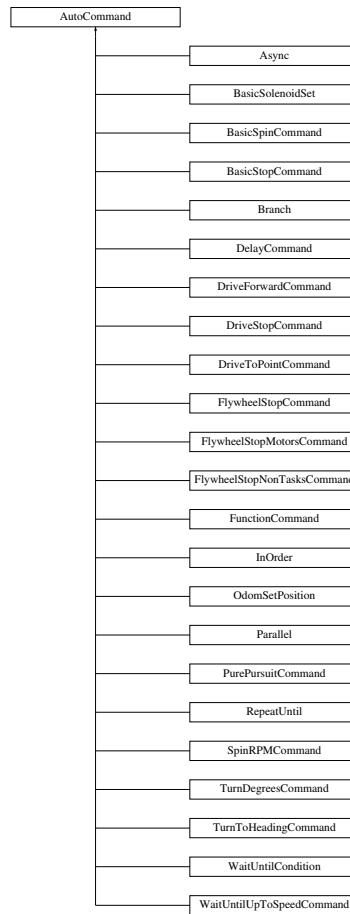
< a list of all possible auto choices

The documentation for this class was generated from the following files:

- [include/utils/auto\\_chooser.h](#)
- [src/utils/auto\\_chooser.cpp](#)

## 5.4 AutoCommand Class Reference

Inheritance diagram for AutoCommand:



### Public Member Functions

- virtual bool `run ()`
- virtual void `on_timeout ()`
- `AutoCommand * withTimeout (double t_seconds)`
- `AutoCommand * withCancelCondition (Condition *true_to_end)`

### Public Attributes

- double `timeout_seconds` = default\_timeout
- `Condition * true_to_end` = nullptr

### Static Public Attributes

- static constexpr double `default_timeout` = 10.0

## 5.4.1 Member Function Documentation

### 5.4.1.1 on\_timeout()

```
virtual void AutoCommand::on_timeout ( ) [inline], [virtual]
```

What to do if we timeout instead of finishing. timeout is specified by the timeout seconds in the constructor

Reimplemented in [InOrder](#), [Parallel](#), [Branch](#), [RepeatUntil](#), [DriveForwardCommand](#), [TurnDegreesCommand](#), [TurnToHeadingCommand](#), [PurePursuitCommand](#), and [DriveStopCommand](#).

### 5.4.1.2 run()

```
virtual bool AutoCommand::run ( ) [inline], [virtual]
```

Executes the command Overridden by child classes

#### Returns

true when the command is finished, false otherwise

Reimplemented in [FunctionCommand](#), [WaitUntilCondition](#), [InOrder](#), [Parallel](#), [Branch](#), [Async](#), [RepeatUntil](#), [BasicSpinCommand](#), [BasicStopCommand](#), [BasicSolenoidSet](#), [DelayCommand](#), [DriveForwardCommand](#), [TurnDegreesCommand](#), [DriveToPointCommand](#), [TurnToHeadingCommand](#), [PurePursuitCommand](#), [DriveStopCommand](#), [OdomSetPosition](#), [SpinRPMCommand](#), [WaitUntilUpToSpeedCommand](#), [FlywheelStopCommand](#), and [FlywheelStopMotorsCommand](#)

## 5.4.2 Member Data Documentation

### 5.4.2.1 timeout\_seconds

```
double AutoCommand::timeout_seconds = default_timeout
```

How long to run until we cancel this command. If the command is cancelled, [on\\_timeout\(\)](#) is called to allow any cleanup from the function. If the timeout\_seconds <= 0, no timeout will be applied and this command will run forever. A timeout can come in handy for some commands that can not reach the end due to some physical limitation such as

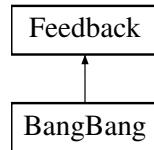
- a drive command hitting a wall and not being able to reach its target
- a command that waits until something is up to speed that never gets up to speed because of battery voltage
- something else...

The documentation for this class was generated from the following file:

- include/utils/command\_structure/auto\_command.h

## 5.5 BangBang Class Reference

Inheritance diagram for BangBang:



### Public Member Functions

- **BangBang** (double threshold, double low, double high)
- void **init** (double start\_pt, double set\_pt, double start\_vel=0.0, double end\_vel=0.0) override
- double **update** (double val) override
- double **get** () override
- void **set\_limits** (double lower, double upper) override
- bool **is\_on\_target** () override

#### 5.5.1 Member Function Documentation

##### 5.5.1.1 get()

```
double BangBang::get () [override], [virtual]
```

###### Returns

the last saved result from the feedback controller

Implements [Feedback](#).

##### 5.5.1.2 init()

```
void BangBang::init (
    double start_pt,
    double set_pt,
    double start_vel = 0.0,
    double end_vel = 0.0 ) [override], [virtual]
```

Initialize the feedback controller for a movement

###### Parameters

<i>start_pt</i>	the current sensor value
<i>set_pt</i>	where the sensor value should be
<i>start_vel</i>	Movement starting velocity
<i>end_vel</i>	Movement ending velocity

Implements [Feedback](#).

#### 5.5.1.3 `is_on_target()`

```
bool BangBang::is_on_target ( ) [override], [virtual]
```

##### Returns

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

#### 5.5.1.4 `set_limits()`

```
void BangBang::set_limits (
    double lower,
    double upper ) [override], [virtual]
```

Clamp the upper and lower limits of the output. If both are 0, no limits should be applied.

##### Parameters

<i>lower</i>	Upper limit
<i>upper</i>	Lower limit

Implements [Feedback](#).

#### 5.5.1.5 `update()`

```
double BangBang::update (
    double val ) [override], [virtual]
```

Iterate the feedback loop once with an updated sensor value

##### Parameters

<i>val</i>	value from the sensor
------------	-----------------------

##### Returns

feedback loop result

Implements [Feedback](#).

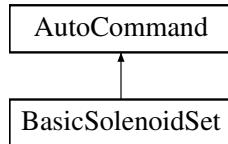
The documentation for this class was generated from the following files:

- include/utils/controls/bang\_bang.h
- src/utils/controls/bang\_bang.cpp

## 5.6 BasicSolenoidSet Class Reference

```
#include <basic_command.h>
```

Inheritance diagram for BasicSolenoidSet:



### Public Member Functions

- **BasicSolenoidSet** (vex::pneumatics &solenoid, bool setting)  
*Construct a new BasicSolenoidSet Command.*
- bool **run** () override  
*Runs the BasicSolenoidSet Overrides run command from AutoCommand.*

### Public Member Functions inherited from AutoCommand

- virtual void **on\_timeout** ()
- **AutoCommand** \* **withTimeout** (double t\_seconds)
- **AutoCommand** \* **withCancelCondition** (**Condition** \*true\_to\_end)

### Additional Inherited Members

#### Public Attributes inherited from AutoCommand

- double **timeout\_seconds** = default\_timeout
- **Condition** \* **true\_to\_end** = nullptr

#### Static Public Attributes inherited from AutoCommand

- static constexpr double **default\_timeout** = 10.0

### 5.6.1 Detailed Description

[AutoCommand](#) wrapper class for [BasicSolenoidSet](#) Using the Vex hardware functions

### 5.6.2 Constructor & Destructor Documentation

#### 5.6.2.1 BasicSolenoidSet()

```
BasicSolenoidSet::BasicSolenoidSet (
    vex::pneumatics & solenoid,
    bool setting )
```

Construct a new [BasicSolenoidSet](#) Command.

**Parameters**

<i>solenoid</i>	Solenoid being set
<i>setting</i>	Setting of the solenoid in boolean (true,false)

**5.6.3 Member Function Documentation****5.6.3.1 run()**

```
bool BasicSolenoidSet::run ( ) [override], [virtual]
```

Runs the [BasicSolenoidSet](#) Overrides run command from [AutoCommand](#).

**Returns**

True Command runs once

Reimplemented from [AutoCommand](#).

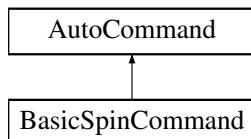
The documentation for this class was generated from the following files:

- include/utils/command\_structure/basic\_command.h
- src/utils/command\_structure/basic\_command.cpp

**5.7 BasicSpinCommand Class Reference**

```
#include <basic_command.h>
```

Inheritance diagram for BasicSpinCommand:

**Public Types**

- enum **type** { **percent** , **voltage** , **veocity** }

**Public Member Functions**

- [\*\*BasicSpinCommand\*\*](#) (vex::motor &motor, vex::directionType dir, BasicSpinCommand::type setting, double power)

*Construct a new BasicSpinCommand.*

- bool [\*\*run\*\*](#) () override

*Runs the BasicSpinCommand Overrides run from Auto Command.*

## Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout \(\)](#)
- [AutoCommand \\* withTimeout](#) (double t\_seconds)
- [AutoCommand \\* withCancelCondition](#) ([Condition](#) \*true\_to\_end)

## Additional Inherited Members

## Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition \\* true\\_to\\_end](#) = nullptr

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

### 5.7.1 Detailed Description

[AutoCommand](#) wrapper class for [BasicSpinCommand](#) using the vex hardware functions

### 5.7.2 Constructor & Destructor Documentation

#### 5.7.2.1 [BasicSpinCommand\(\)](#)

```
BasicSpinCommand::BasicSpinCommand (
    vex::motor & motor,
    vex::directionType dir,
    BasicSpinCommand::type setting,
    double power )
```

Construct a new [BasicSpinCommand](#).

a BasicMotorSpin Command

#### Parameters

<i>motor</i>	Motor to spin
<i>direc</i>	Direction of motor spin
<i>setting</i>	Power setting in volts,percentage,velocity
<i>power</i>	Value of desired power
<i>motor</i>	Motor port to spin
<i>dir</i>	Direction for spinning
<i>setting</i>	Power setting in volts,percentage,velocity
<i>power</i>	Value of desired power

### 5.7.3 Member Function Documentation

#### 5.7.3.1 run()

```
bool BasicSpinCommand::run ( ) [override], [virtual]
```

Runs the [BasicSpinCommand](#) Overrides run from Auto Command.

Run the [BasicSpinCommand](#) Overrides run from Auto Command.

##### Returns

True [Async](#) running command

True Command runs once

Reimplemented from [AutoCommand](#).

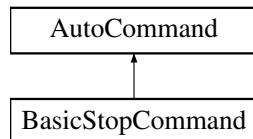
The documentation for this class was generated from the following files:

- include/utils/command\_structure/basic\_command.h
- src/utils/command\_structure/basic\_command.cpp

## 5.8 BasicStopCommand Class Reference

```
#include <basic_command.h>
```

Inheritance diagram for BasicStopCommand:



### Public Member Functions

- [BasicStopCommand](#) (vex::motor &motor, vex::brakeType setting)  
*Construct a new BasicMotorStop Command.*
- bool [run \(\)](#) override  
*Runs the BasicMotorStop Command Overrides run command from [AutoCommand](#).*

### Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout \(\)](#)
- [AutoCommand \\* withTimeout](#) (double t\_seconds)
- [AutoCommand \\* withCancelCondition](#) ([Condition](#) \*true\_to\_end)

## Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- double `timeout_seconds` = `default_timeout`
- `Condition * true_to_end` = `nullptr`

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double `default_timeout` = 10.0

## 5.8.1 Detailed Description

[AutoCommand](#) wrapper class for [BasicStopCommand](#) Using the Vex hardware functions

## 5.8.2 Constructor & Destructor Documentation

### 5.8.2.1 [BasicStopCommand\(\)](#)

```
BasicStopCommand::BasicStopCommand (
    vex::motor & motor,
    vex::brakeType setting )
```

Construct a new BasicMotorStop Command.

Construct a BasicMotorStop Command.

#### Parameters

<code>motor</code>	The motor to stop
<code>setting</code>	The brake setting for the motor
<code>motor</code>	Motor to stop
<code>setting</code>	Braketype setting brake,coast,hold

## 5.8.3 Member Function Documentation

### 5.8.3.1 [run\(\)](#)

```
bool BasicStopCommand::run ( ) [override], [virtual]
```

Runs the BasicMotorStop Command Overrides run command from [AutoCommand](#).

Runs the BasicMotorStop command Ovverides run command from [AutoCommand](#).

**Returns**

True Command runs once

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

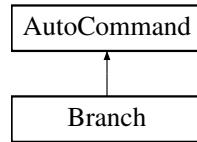
- include/utils/command\_structure/basic\_command.h
- src/utils/command\_structure/basic\_command.cpp

## 5.9 Branch Class Reference

[Branch](#) chooses from multiple options at runtime. the function decider returns an index into the choices vector If you wish to make no choice and skip this section, return NO\_CHOICE; any choice that is out of bounds set to NO\_CHOICE.

```
#include <auto_command.h>
```

Inheritance diagram for Branch:



### Public Member Functions

- **Branch** ([Condition](#) \*cond, [AutoCommand](#) \*false\_choice, [AutoCommand](#) \*true\_choice)
- bool [run](#) () override
- void [on\\_timeout](#) () override

### Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

### Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

### 5.9.1 Detailed Description

[Branch](#) chooses from multiple options at runtime. the function decider returns an index into the choices vector If you wish to make no choice and skip this section, return NO\_CHOICE; any choice that is out of bounds set to NO\_CHOICE.

### 5.9.2 Member Function Documentation

#### 5.9.2.1 on\_timeout()

```
void Branch::on_timeout ( ) [override], [virtual]
```

What to do if we timeout instead of finishing. timeout is specified by the timeout seconds in the constructor

Reimplemented from [AutoCommand](#).

#### 5.9.2.2 run()

```
bool Branch::run ( ) [override], [virtual]
```

Executes the command Overridden by child classes

##### Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command\_structure/auto\_command.h
- src/utils/command\_structure/auto\_command.cpp

## 5.10 screen::ButtonWidget Class Reference

Widget that does something when you tap it. The function is only called once when you first tap it.

```
#include <screen.h>
```

### Public Member Functions

- [ButtonWidget](#) (std::function< void(void)> onpress, [Rect](#) rect, std::string name)
   
*Create a Button widget.*
- [ButtonWidget](#) (void(\*onpress)(), [Rect](#) rect, std::string name)
   
*Create a Button widget.*
- bool [update](#) (bool was\_pressed, int x, int y)
   
*responds to user input*
- void [draw](#) (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number)
   
*draws the button to the screen*

### 5.10.1 Detailed Description

Widget that does something when you tap it. The function is only called once when you first tap it.

### 5.10.2 Constructor & Destructor Documentation

#### 5.10.2.1 ButtonWidget() [1/2]

```
screen::ButtonWidget::ButtonWidget (
    std::function< void(void)> onpress,
    Rect rect,
    std::string name ) [inline]
```

Create a Button widget.

##### Parameters

<i>onpress</i>	the function to be called when the button is tapped
<i>rect</i>	the area the button should take up on the screen
<i>name</i>	the label put on the button

#### 5.10.2.2 ButtonWidget() [2/2]

```
screen::ButtonWidget::ButtonWidget (
    void(*)() onpress,
    Rect rect,
    std::string name ) [inline]
```

Create a Button widget.

##### Parameters

<i>onpress</i>	the function to be called when the button is tapped
<i>rect</i>	the area the button should take up on the screen
<i>name</i>	the label put on the button

### 5.10.3 Member Function Documentation

#### 5.10.3.1 update()

```
bool screen::ButtonWidget::update (
    bool was_pressed,
    int x,
    int y )
```

responds to user input

**Parameters**

<code>was_pressed</code>	if the screen is pressed
<code>x</code>	x position if the screen was pressed
<code>y</code>	y position if the screen was pressed

**Returns**

true if the button was pressed

The documentation for this class was generated from the following files:

- include/subsystems/screen.h
- src/subsystems/screen.cpp

## 5.11 CommandController Class Reference

```
#include <command_controller.h>
```

### Public Member Functions

- **CommandController ()**  
*Create an empty `CommandController`. Add Command with `CommandController::add()`*
- **CommandController (std::initializer\_list< AutoCommand \* > cmd)**  
*Create a `CommandController` with commands pre added. More can be added with `CommandController::add()`*
- void **add (std::vector< AutoCommand \* > cmd)**
- void **add (AutoCommand \*cmd, double timeout\_seconds=10.0)**
- void **add (std::vector< AutoCommand \* > cmd, double timeout\_sec)**
- void **add\_delay (int ms)**
- void **add\_cancel\_func (std::function< bool(void)> true\_if\_cancel)**  
*add\_cancel\_func specifies that when this func evaluates to true, to cancel the command controller*
- void **run ()**
- bool **last\_command\_timed\_out ()**

### 5.11.1 Detailed Description

File: `command_controller.h` Desc: A `CommandController` manages the AutoCommands that make up an autonomous route. The AutoCommands are kept in a queue and get executed and removed from the queue in FIFO order.

### 5.11.2 Constructor & Destructor Documentation

#### 5.11.2.1 CommandController()

```
CommandController::CommandController (
    std::initializer_list< AutoCommand * > cmd ) [inline]
```

Create a `CommandController` with commands pre added. More can be added with `CommandController::add()`

## Parameters

<i>cmds</i>	
-------------	--

**5.11.3 Member Function Documentation****5.11.3.1 add() [1/3]**

```
void CommandController::add (
    AutoCommand * cmd,
    double timeout_seconds = 10.0 )
```

File: command\_controller.cpp Desc: A [CommandController](#) manages the AutoCommands that make up an autonomous route. The AutoCommands are kept in a queue and get executed and removed from the queue in FIFO order. Adds a command to the queue

## Parameters

<i>cmd</i>	the <a href="#">AutoCommand</a> we want to add to our list
<i>timeout_seconds</i>	the number of seconds we will let the command run for. If it exceeds this, we cancel it and run on_timeout

**5.11.3.2 add() [2/3]**

```
void CommandController::add (
    std::vector< AutoCommand * > cmds )
```

Adds a command to the queue

## Parameters

<i>cmd</i>	the <a href="#">AutoCommand</a> we want to add to our list
<i>timeout_seconds</i>	the number of seconds we will let the command run for. If it exceeds this, we cancel it and run on_timeout. if it is <= 0 no time out will be applied

Add multiple commands to the queue. No timeout here.

## Parameters

<i>cmds</i>	the AutoCommands we want to add to our list
-------------	---

**5.11.3.3 add() [3/3]**

```
void CommandController::add (
    std::vector< AutoCommand * > cmds,
    double timeout_sec )
```

Add multiple commands to the queue. No timeout here.

**Parameters**

<i>cmds</i>	the AutoCommands we want to add to our list Add multiple commands to the queue. No timeout here.
<i>cmds</i>	the AutoCommands we want to add to our list
<i>timeout_sec</i>	timeout in seconds to apply to all commands if they are still the default

Add multiple commands to the queue. No timeout here.

**Parameters**

<i>cmds</i>	the AutoCommands we want to add to our list
<i>timeout</i>	timeout in seconds to apply to all commands if they are still the default

**5.11.3.4 add\_cancel\_func()**

```
void CommandController::add_cancel_func (
    std::function< bool(void)> true_if_cancel )
```

add\_cancel\_func specifies that when this func evaluates to true, to cancel the command controller

**Parameters**

<i>true_if_cancel</i>	a function that returns true when we want to cancel the command controller
-----------------------	--

**5.11.3.5 add\_delay()**

```
void CommandController::add_delay (
    int ms )
```

Adds a command that will delay progression of the queue

**Parameters**

<i>ms</i>	- number of milliseconds to wait before continuing execution of autonomous
-----------	--

**5.11.3.6 last\_command\_timed\_out()**

```
bool CommandController::last_command_timed_out ( )
```

last\_command\_timed\_out tells how the last command ended Use this if you want to make decisions based on the end of the last command

**Returns**

true if the last command timed out. false if it finished regularly

### 5.11.3.7 run()

```
void CommandController::run ( )
```

Begin execution of the queue Execute and remove commands in FIFO order

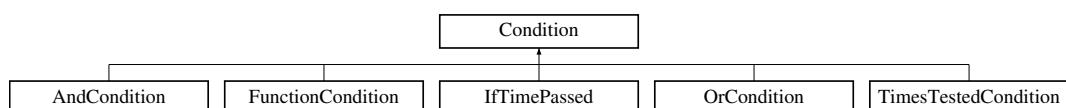
The documentation for this class was generated from the following files:

- include/utils/command\_structure/command\_controller.h
- src/utils/command\_structure/command\_controller.cpp

## 5.12 Condition Class Reference

```
#include <auto_command.h>
```

Inheritance diagram for Condition:



### Public Member Functions

- **Condition \* Or (Condition \*b)**
- **Condition \* And (Condition \*b)**
- virtual bool **test ()=0**

### 5.12.1 Detailed Description

File: [auto\\_command.h](#) Desc: Interface for module-specific commands A **Condition** is a function that returns true or false is\_even is a predicate that would return true if a number is even For our purposes, a **Condition** is a choice to be made at runtime drive\_sys.reached\_point(10, 30) is a predicate time.has\_elapsed(10, vex::seconds) is a predicate extend this class for different choices you wish to make

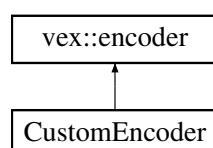
The documentation for this class was generated from the following files:

- include/utils/command\_structure/auto\_command.h
- src/utils/command\_structure/auto\_command.cpp

## 5.13 CustomEncoder Class Reference

```
#include <custom_encoder.h>
```

Inheritance diagram for CustomEncoder:



## Public Member Functions

- `CustomEncoder (vex::triport::port &port, double ticks_per_rev)`
- `void setRotation (double val, vex::rotationUnits units)`
- `void setPosition (double val, vex::rotationUnits units)`
- `double rotation (vex::rotationUnits units)`
- `double position (vex::rotationUnits units)`
- `double velocity (vex::velocityUnits units)`

### 5.13.1 Detailed Description

A wrapper class for the vex encoder that allows the use of 3rd party encoders with different tick-per-revolution values.

### 5.13.2 Constructor & Destructor Documentation

#### 5.13.2.1 CustomEncoder()

```
CustomEncoder::CustomEncoder (
    vex::triport::port & port,
    double ticks_per_rev )
```

Construct an encoder with a custom number of ticks

##### Parameters

<i>port</i>	the tripot port on the brain the encoder is plugged into
<i>ticks_per_rev</i>	the number of ticks the encoder will report for one revolution

### 5.13.3 Member Function Documentation

#### 5.13.3.1 position()

```
double CustomEncoder::position (
    vex::rotationUnits units )
```

get the position that the encoder is at

##### Parameters

<i>units</i>	the unit we want the return value to be in
--------------	--

##### Returns

the position of the encoder in the units specified

### 5.13.3.2 rotation()

```
double CustomEncoder::rotation (
    vex::rotationUnits units )
```

get the rotation that the encoder is at

#### Parameters

<i>units</i>	the unit we want the return value to be in
--------------	--

#### Returns

the rotation of the encoder in the units specified

### 5.13.3.3 setPosition()

```
void CustomEncoder::setPosition (
    double val,
    vex::rotationUnits units )
```

sets the stored position of the encoder. Any further movements will be from this value

#### Parameters

<i>val</i>	the numerical value of the position we are setting to
<i>units</i>	the unit of val

### 5.13.3.4 setRotation()

```
void CustomEncoder::setRotation (
    double val,
    vex::rotationUnits units )
```

sets the stored rotation of the encoder. Any further movements will be from this value

#### Parameters

<i>val</i>	the numerical value of the angle we are setting to
<i>units</i>	the unit of val

### 5.13.3.5 velocity()

```
double CustomEncoder::velocity (
    vex::velocityUnits units )
```

get the velocity that the encoder is moving at

**Parameters**

<i>units</i>	the unit we want the return value to be in
--------------	--

**Returns**

the velocity of the encoder in the units specified

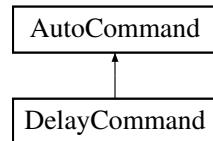
The documentation for this class was generated from the following files:

- include/subsystems/custom\_encoder.h
- src/subsystems/custom\_encoder.cpp

## 5.14 DelayCommand Class Reference

```
#include <delay_command.h>
```

Inheritance diagram for DelayCommand:



### Public Member Functions

- [DelayCommand](#) (int ms)
- bool [run](#) () override

### Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

### Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

### 5.14.1 Detailed Description

File: [delay\\_command.h](#) Desc: A [DelayCommand](#) will make the robot wait the set amount of milliseconds before continuing execution of the autonomous route

### 5.14.2 Constructor & Destructor Documentation

#### 5.14.2.1 DelayCommand()

```
DelayCommand::DelayCommand (
    int ms ) [inline]
```

Construct a delay command

Parameters

<i>ms</i>	the number of milliseconds to delay for
-----------	---

### 5.14.3 Member Function Documentation

#### 5.14.3.1 run()

```
bool DelayCommand::run ( ) [inline], [override], [virtual]
```

Delays for the amount of milliseconds stored in the command Overrides run from [AutoCommand](#)

Returns

true when complete

Reimplemented from [AutoCommand](#).

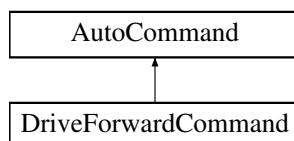
The documentation for this class was generated from the following file:

- include/utils/command\_structure/delay\_command.h

## 5.15 DriveForwardCommand Class Reference

```
#include <drive_commands.h>
```

Inheritance diagram for DriveForwardCommand:



### Public Member Functions

- `DriveForwardCommand (TankDrive &drive_sys, Feedback &feedback, double inches, directionType dir, double max_speed=1, double end_speed=0)`
- `bool run () override`
- `void on_timeout () override`

### Public Member Functions inherited from [AutoCommand](#)

- `AutoCommand * withTimeout (double t_seconds)`
- `AutoCommand * withCancelCondition (Condition *true_to_end)`

### Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- `double timeout_seconds = default_timeout`
- `Condition * true_to_end = nullptr`

### Static Public Attributes inherited from [AutoCommand](#)

- `static constexpr double default_timeout = 10.0`

## 5.15.1 Detailed Description

[AutoCommand](#) wrapper class for the drive\_forward function in the [TankDrive](#) class

## 5.15.2 Constructor & Destructor Documentation

### 5.15.2.1 DriveForwardCommand()

```
DriveForwardCommand::DriveForwardCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    double inches,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

File: [drive\\_commands.h](#) Desc: Holds all the [AutoCommand](#) subclasses that wrap (currently) [TankDrive](#) functions

Currently includes:

- `drive_forward`
- `turn_degrees`
- `drive_to_point`
- `turn_to_heading`
- `stop`

Also holds [AutoCommand](#) subclasses that wrap [OdometryBase](#) functions

Currently includes:

- `set_position` Construct a DriveForward Command

**Parameters**

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the drive
<i>inches</i>	how far forward to drive
<i>dir</i>	the direction to drive
<i>max_speed</i>	0 -> 1 percentage of the drive systems speed to drive at

**5.15.3 Member Function Documentation****5.15.3.1 on\_timeout()**

```
void DriveForwardCommand::on_timeout ( ) [override], [virtual]
```

Cleans up drive system if we time out before finishing

reset the drive system if we timeout

Reimplemented from [AutoCommand](#).

**5.15.3.2 run()**

```
bool DriveForwardCommand::run ( ) [override], [virtual]
```

Run drive\_forward Overrides run from [AutoCommand](#)

**Returns**

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

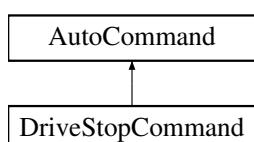
The documentation for this class was generated from the following files:

- include/utils/command\_structure/drive\_commands.h
- src/utils/command\_structure/drive\_commands.cpp

**5.16 DriveStopCommand Class Reference**

```
#include <drive_commands.h>
```

Inheritance diagram for DriveStopCommand:



## Public Member Functions

- `DriveStopCommand (TankDrive &drive_sys)`
- `bool run () override`
- `void on_timeout () override`

## Public Member Functions inherited from [AutoCommand](#)

- `AutoCommand * withTimeout (double t_seconds)`
- `AutoCommand * withCancelCondition (Condition *true_to_end)`

## Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- `double timeout_seconds = default_timeout`
- `Condition * true_to_end = nullptr`

### Static Public Attributes inherited from [AutoCommand](#)

- `static constexpr double default_timeout = 10.0`

## 5.16.1 Detailed Description

[AutoCommand](#) wrapper class for the stop() function in the [TankDrive](#) class

## 5.16.2 Constructor & Destructor Documentation

### 5.16.2.1 [DriveStopCommand\(\)](#)

```
DriveStopCommand::DriveStopCommand (
    TankDrive & drive_sys )
```

Construct a DriveStop Command

#### Parameters

<code>drive_sys</code>	the drive system we are commanding
------------------------	------------------------------------

## 5.16.3 Member Function Documentation

### 5.16.3.1 [on\\_timeout\(\)](#)

```
void DriveStopCommand::on_timeout ( ) [override], [virtual]
```

What to do if we timeout instead of finishing. timeout is specified by the timeout seconds in the constructor

Reimplemented from [AutoCommand](#).

### 5.16.3.2 run()

```
bool DriveStopCommand::run ( ) [override], [virtual]
```

Stop the drive system Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

Stop the drive train Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

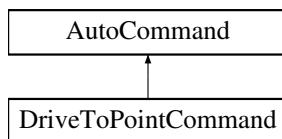
The documentation for this class was generated from the following files:

- include/utils/command\_structure/drive\_commands.h
- src/utils/command\_structure/drive\_commands.cpp

## 5.17 DriveToPointCommand Class Reference

```
#include <drive_commands.h>
```

Inheritance diagram for DriveToPointCommand:



### Public Member Functions

- [DriveToPointCommand](#) ([TankDrive](#) &drive\_sys, [Feedback](#) &feedback, double x, double y, directionType dir, double max\_speed=1, double end\_speed=0)
- [DriveToPointCommand](#) ([TankDrive](#) &drive\_sys, [Feedback](#) &feedback, [point\\_t](#) point, directionType dir, double max\_speed=1, double end\_speed=0)
- bool [run](#) () override

## Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand \\* withTimeout](#) (double t\_seconds)
- [AutoCommand \\* withCancelCondition](#) ([Condition](#) \*true\_to\_end)

## Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

## 5.17.1 Detailed Description

[AutoCommand](#) wrapper class for the drive\_to\_point function in the [TankDrive](#) class

## 5.17.2 Constructor & Destructor Documentation

### 5.17.2.1 DriveToPointCommand() [1/2]

```
DriveToPointCommand::DriveToPointCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    double x,
    double y,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

Construct a DriveForward Command

#### Parameters

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the drive
<i>x</i>	where to drive in the x dimension
<i>y</i>	where to drive in the y dimension
<i>dir</i>	the direction to drive
<i>max_speed</i>	0 -> 1 percentage of the drive systems speed to drive at

### 5.17.2.2 DriveToPointCommand() [2/2]

```
DriveToPointCommand::DriveToPointCommand (
```

```

    TankDrive & drive_sys,
    Feedback & feedback,
    point_t point,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0 )

```

Construct a DriveForward Command

#### Parameters

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the drive
<i>point</i>	the point to drive to
<i>dir</i>	the direction to drive
<i>max_speed</i>	0 -> 1 percentage of the drive systems speed to drive at

### 5.17.3 Member Function Documentation

#### 5.17.3.1 run()

```
bool DriveToPointCommand::run ( ) [override], [virtual]
```

Run drive\_to\_point Overrides run from [AutoCommand](#)

#### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command\_structure/drive\_commands.h
- src/utils/command\_structure/drive\_commands.cpp

### 5.18 AutoChooser::entry\_t Struct Reference

```
#include <auto_chooser.h>
```

#### Public Attributes

- [Rect rect](#)
- std::string [name](#)

#### 5.18.1 Detailed Description

[entry\\_t](#) is a datatype used to store information that the chooser knows about an auto selection button

## 5.18.2 Member Data Documentation

### 5.18.2.1 name

`std::string AutoChooser::entry_t::name`

name of the auto repretsented by the block

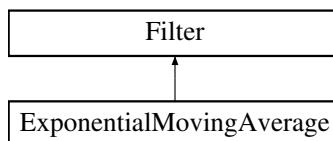
The documentation for this struct was generated from the following file:

- `include/utils/auto_chooser.h`

## 5.19 ExponentialMovingAverage Class Reference

`#include <moving_average.h>`

Inheritance diagram for ExponentialMovingAverage:



### Public Member Functions

- [ExponentialMovingAverage \(int buffer\\_size\)](#)
- [ExponentialMovingAverage \(int buffer\\_size, double starting\\_value\)](#)
- void [add\\_entry \(double n\)](#)
- double [get\\_value \(\)](#)
- int [get\\_size \(\)](#)

### Public Member Functions inherited from [Filter](#)

- virtual double [get\\_value \(\) const =0](#)

## 5.19.1 Detailed Description

### [ExponentialMovingAverage](#)

An exponential moving average is a way of smoothing out noisy data. For many sensor readings, the noise is roughly symmetric around the actual value. This means that if you collect enough samples those that are too high are cancelled out by the samples that are too low leaving the real value.

A simple mobing average lags significantly with time as it has to counteract old samples. An exponential moving average keeps more up to date by weighting newer readings higher than older readings so it is more up to date while also still smoothed.

The [ExponentialMovingAverage](#) class provides an simple interface to do this smoothing from our noisy sensor values.

## 5.19.2 Constructor & Destructor Documentation

### 5.19.2.1 [ExponentialMovingAverage\(\) \[1/2\]](#)

```
ExponentialMovingAverage::ExponentialMovingAverage (
    int buffer_size )
```

Create a moving average calculator with 0 as the default value

**Parameters**

<i>buffer_size</i>	The size of the buffer. The number of samples that constitute a valid reading
--------------------	---

**5.19.2.2 ExponentialMovingAverage() [2/2]**

```
ExponentialMovingAverage::ExponentialMovingAverage (
    int buffer_size,
    double starting_value )
```

Create a moving average calculator with a specified default value

**Parameters**

<i>buffer_size</i>	The size of the buffer. The number of samples that constitute a valid reading
<i>starting_value</i>	The value that the average will be before any data is added

**5.19.3 Member Function Documentation****5.19.3.1 add\_entry()**

```
void ExponentialMovingAverage::add_entry (
    double n) [virtual]
```

Add a reading to the buffer Before: [ 1 1 2 2 3 3 ] => 2 ^ After: [ 2 1 2 2 3 3 ] => 2.16 ^

**Parameters**

<i>n</i>	the sample that will be added to the moving average.
----------	--

Implements [Filter](#).

**5.19.3.2 get\_size()**

```
int ExponentialMovingAverage::get_size ( )
```

How many samples the average is made from

**Returns**

the number of samples used to calculate this average

### 5.19.3.3 `get_value()`

```
double ExponentialMovingAverage::get_value ( )
```

Returns the average based off of all the samples collected so far

#### Returns

the calculated average.  $\text{sum(samples)}/\text{numsamples}$

How many samples the average is made from

#### Returns

the number of samples used to calculate this average

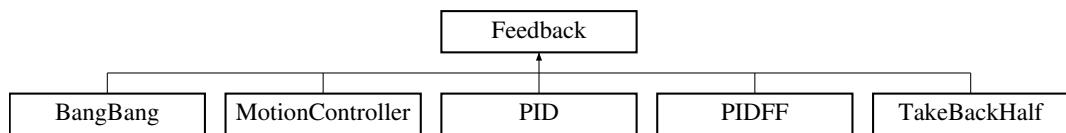
The documentation for this class was generated from the following files:

- `include/utils/moving_average.h`
- `src/utils/moving_average.cpp`

## 5.20 Feedback Class Reference

```
#include <feedback_base.h>
```

Inheritance diagram for Feedback:



### Public Member Functions

- virtual void `init` (double start\_pt, double set\_pt, double start\_vel=0.0, double end\_vel=0.0)=0
- virtual double `update` (double val)=0
- virtual double `get` ()=0
- virtual void `set_limits` (double lower, double upper)=0
- virtual bool `is_on_target` ()=0

### 5.20.1 Detailed Description

Interface so that subsystems can easily switch between feedback loops

#### Author

Ryan McGee

#### Date

9/25/2022

## 5.20.2 Member Function Documentation

### 5.20.2.1 get()

```
virtual double Feedback::get ( ) [pure virtual]
```

#### Returns

the last saved result from the feedback controller

Implemented in [BangBang](#), [MotionController](#), [PID](#), [PIDFF](#), and [TakeBackHalf](#).

### 5.20.2.2 init()

```
virtual void Feedback::init (
    double start_pt,
    double set_pt,
    double start_vel = 0.0,
    double end_vel = 0.0 ) [pure virtual]
```

Initialize the feedback controller for a movement

#### Parameters

<i>start_pt</i>	the current sensor value
<i>set_pt</i>	where the sensor value should be
<i>start_vel</i>	Movement starting velocity
<i>end_vel</i>	Movement ending velocity

Implemented in [MotionController](#), [PIDFF](#), [PID](#), [BangBang](#), and [TakeBackHalf](#).

### 5.20.2.3 is\_on\_target()

```
virtual bool Feedback::is_on_target ( ) [pure virtual]
```

#### Returns

true if the feedback controller has reached it's setpoint

Implemented in [BangBang](#), [MotionController](#), [PID](#), [PIDFF](#), and [TakeBackHalf](#).

### 5.20.2.4 set\_limits()

```
virtual void Feedback::set_limits (
    double lower,
    double upper ) [pure virtual]
```

Clamp the upper and lower limits of the output. If both are 0, no limits should be applied.

**Parameters**

<i>lower</i>	Upper limit
<i>upper</i>	Lower limit

Implemented in [BangBang](#), [MotionController](#), [PID](#), [PIDFF](#), and [TakeBackHalf](#).

### 5.20.2.5 update()

```
virtual double Feedback::update (
    double val ) [pure virtual]
```

Iterate the feedback loop once with an updated sensor value

**Parameters**

<i>val</i>	value from the sensor
------------	-----------------------

**Returns**

feedback loop result

Implemented in [MotionController](#), [PID](#), [BangBang](#), [PIDFF](#), and [TakeBackHalf](#).

The documentation for this class was generated from the following file:

- include/utils/controls/feedback\_base.h

## 5.21 FeedForward Class Reference

```
#include <feedforward.h>
```

**Classes**

- struct [ff\\_config\\_t](#)

**Public Member Functions**

- [FeedForward \(ff\\_config\\_t &cfg\)](#)
- double [calculate](#) (double v, double a, double pid\_ref=0.0)  
*Perform the feedforward calculation.*

### 5.21.1 Detailed Description

#### FeedForward

Stores the feedforward constants, and allows for quick computation. Feedforward should be used in systems that require smooth precise movements and have high inertia, such as drivetrains and lifts.

This is best used alongside a **PID** loop, with the form: `output = pid.get() + feedforward.calculate(v, a);`

In this case, the feedforward does the majority of the heavy lifting, and the pid loop only corrects for inconsistencies

For information about tuning feedforward, I recommend looking at this post: <https://www.chiefdelphi.com/t/paper-frc-drivetrain-characterization/160915> (yes I know it's for FRC but trust me, it's useful)

#### Author

Ryan McGee

#### Date

6/13/2022

### 5.21.2 Constructor & Destructor Documentation

#### 5.21.2.1 FeedForward()

```
FeedForward::FeedForward (
    ff_config_t & cfg ) [inline]
```

Creates a **FeedForward** object.

#### Parameters

<code>cfg</code>	Configuration Struct for tuning
------------------	---------------------------------

### 5.21.3 Member Function Documentation

#### 5.21.3.1 calculate()

```
double FeedForward::calculate (
    double v,
    double a,
    double pid_ref = 0.0 ) [inline]
```

Perform the feedforward calculation.

This calculation is the equation:  $F = kG + kS * \text{sgn}(v) + kV * v + kA * a$

**Parameters**

<i>v</i>	Requested velocity of system
<i>a</i>	Requested acceleration of system

**Returns**

A feedforward that should closely represent the system if tuned correctly

The documentation for this class was generated from the following file:

- include/utils/controls/feedforward.h

## 5.22 FeedForward::ff\_config\_t Struct Reference

```
#include <feedforward.h>
```

**Public Attributes**

- double kS
- double kV
- double kA
- double kG

### 5.22.1 Detailed Description

`ff_config_t` holds the parameters to make the theoretical model of a real world system equation is of the form kS if the system is not stopped, 0 otherwise

- kV \* desired velocity
- kA \* desired acceleration
- kG

### 5.22.2 Member Data Documentation

#### 5.22.2.1 kA

```
double FeedForward::ff_config_t::kA
```

kA - Acceleration coefficient: the power required to change the mechanism's speed. Multiplied by the requested acceleration.

### 5.22.2.2 kG

```
double FeedForward::ff_config_t::kG
```

kG - Gravity coefficient: only needed for lifts. The power required to overcome gravity and stay at steady state.

### 5.22.2.3 kS

```
double FeedForward::ff_config_t::kS
```

Coefficient to overcome static friction: the point at which the motor *starts* to move.

### 5.22.2.4 kV

```
double FeedForward::ff_config_t::kV
```

Velocity coefficient: the power required to keep the mechanism in motion. Multiplied by the requested velocity.

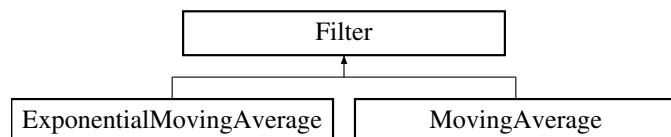
The documentation for this struct was generated from the following file:

- include/utils/controls/feedforward.h

## 5.23 Filter Class Reference

```
#include <moving_average.h>
```

Inheritance diagram for Filter:



### Public Member Functions

- virtual void [add\\_entry](#) (double n)=0
- virtual double [get\\_value](#) () const =0

### 5.23.1 Detailed Description

Interface for filters Use add\_entry to supply data and get\_value to retrieve the filtered value

## 5.23.2 Member Function Documentation

### 5.23.2.1 add\_entry()

```
virtual void Filter::add_entry (
    double n ) [pure virtual]
```

Implemented in [MovingAverage](#), and [ExponentialMovingAverage](#).

### 5.23.2.2 get\_value()

```
virtual double Filter::get_value ( ) const [pure virtual]
```

Implemented in [MovingAverage](#).

The documentation for this class was generated from the following file:

- include/utils/moving\_average.h

## 5.24 Flywheel Class Reference

```
#include <flywheel.h>
```

### Public Member Functions

- [Flywheel](#) (vex::motor\_group &motors, [Feedback](#) &feedback, [FeedForward](#) &helper, const double ratio, [Filter](#) &filt)
- double [get\\_target](#) () const
- double [getRPM](#) () const
- vex::motor\_group & [get\\_motors](#) () const
- void [spin\\_manual](#) (double speed, directionType dir=fwd)
- void [spin\\_rpm](#) (double rpm)
- void [stop](#) ()
- bool [is\\_on\\_target](#) ()
 

*check if the feedback controller thinks the flywheel is on target*
- [screen::Page](#) \* [Page](#) () const
 

*Creates a page displaying info about the flywheel.*
- [AutoCommand](#) \* [SpinRpmCmd](#) (int rpm)
 

*Creates a new auto command to spin the flywheel at the desired velocity.*
- [AutoCommand](#) \* [WaitUntilUpToSpeedCmd](#) ()
 

*Creates a new auto command that will hold until the flywheel has its target as defined by its feedback controller.*

### Friends

- class [FlywheelPage](#)
- int [spinRPMTask](#) (void \*wheelPointer)

### 5.24.1 Detailed Description

a [Flywheel](#) class that handles all control of a high inertia spinning disk. It gives multiple options for what control system to use in order to control wheel velocity and functions alerting the user when the flywheel is up to speed. [Flywheel](#) is a set and forget class. Once you create it you can call `spin_rpm` or `stop` on it at any time and it will take all necessary steps to accomplish this.

### 5.24.2 Constructor & Destructor Documentation

#### 5.24.2.1 Flywheel()

```
Flywheel::Flywheel (
    vex::motor_group & motors,
    Feedback & feedback,
    FeedForward & helper,
    const double ratio,
    Filter & filt )
```

Create the [Flywheel](#) object using [PID](#) + feedforward for control.

#### Parameters

<i>motors</i>	pointer to the motors on the fly wheel
<i>feedback</i>	a feedback controller
<i>helper</i>	a feedforward config (only kV is used) to help the feedback controller along
<i>ratio</i>	ratio of the gears from the motor to the flywheel just multiplies the velocity
<i>filter</i>	the filter to use to smooth noisy motor readings

### 5.24.3 Member Function Documentation

#### 5.24.3.1 get\_motors()

```
motor_group & Flywheel::get_motors ( ) const
```

Returns the motors

#### Returns

the motors used to run the flywheel

#### 5.24.3.2 get\_target()

```
double Flywheel::get_target ( ) const
```

Return the target\_rpm that the flywheel is currently trying to achieve

#### Returns

target\_rpm the target rpm

Return the current value that the target\_rpm should be set to

### 5.24.3.3 getRPM()

```
double Flywheel::getRPM ( ) const
```

return the velocity of the flywheel

### 5.24.3.4 is\_on\_target()

```
bool Flywheel::is_on_target ( ) [inline]
```

check if the feedback controller thinks the flywheel is on target

#### Returns

true if on target

### 5.24.3.5 Page()

```
screen::Page * Flywheel::Page ( ) const
```

Creates a page displaying info about the flywheel.

#### Returns

the page should be used for `screen::start\_screen(screen, {fw.Page()});`

### 5.24.3.6 spin\_manual()

```
void Flywheel::spin_manual (
    double speed,
    directionType dir = fwd )
```

Spin motors using voltage; defaults forward at 12 volts FOR USE BY OPCONTROL AND AUTONOMOUS - this only applies if the target\_rpm thread is not running

#### Parameters

<i>speed</i>	- speed (between -1 and 1) to set the motor
<i>dir</i>	- direction that the motor moves in; defaults to forward

Spin motors using voltage; defaults forward at 12 volts FOR USE BY OPCONTROL AND AUTONOMOUS - this only applies if the RPM thread is not running

#### Parameters

<i>speed</i>	- speed (between -1 and 1) to set the motor
<i>dir</i>	- direction that the motor moves in; defaults to forward

### 5.24.3.7 spin\_rpm()

```
void Flywheel::spin_rpm (
    double input_rpm )
```

starts or sets the target\_rpm thread at new value what control scheme is dependent on control\_style

#### Parameters

<i>rpm</i>	- the target_rpm we want to spin at
------------	-------------------------------------

starts or sets the RPM thread at new value what control scheme is dependent on control\_style

#### Parameters

<i>input_rpm</i>	- set the current RPM
------------------	-----------------------

### 5.24.3.8 SpinRpmCmd()

```
AutoCommand * Flywheel::SpinRpmCmd (
    int rpm ) [inline]
```

Creates a new auto command to spin the flywheel at the desired velocity.

#### Parameters

<i>rpm</i>	the rpm to spin at
------------	--------------------

#### Returns

an auto command to add to a command controller

### 5.24.3.9 stop()

```
void Flywheel::stop ( )
```

Stops the motors. If manually spinning, this will do nothing just call spin\_mainual(0.0) to send 0 volts

stop the RPM thread and the wheel

### 5.24.3.10 WaitUntilUpToSpeedCmd()

```
AutoCommand * Flywheel::WaitUntilUpToSpeedCmd ( ) [inline]
```

Creates a new auto command that will hold until the flywheel has its target as defined by its feedback controller.

#### Returns

an auto command to add to a command controller

## 5.24.4 Friends And Related Symbol Documentation

### 5.24.4.1 spinRPMTask

```
int spinRPMTask (
    void * wheelPointer ) [friend]
```

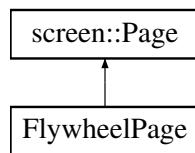
Runs a thread that keeps track of updating flywheel RPM and controlling it accordingly

The documentation for this class was generated from the following files:

- include/subsystems/flywheel.h
- src/subsystems/flywheel.cpp

## 5.25 FlywheelPage Class Reference

Inheritance diagram for FlywheelPage:



### Public Member Functions

- **FlywheelPage** (const [Flywheel](#) &fw)
- void [update](#) (bool, int, int) override
- void [draw](#) (vex::brain::lcd &screen, bool, unsigned int) override

### Static Public Attributes

- static const size\_t [window\\_size](#) = 40

## 5.25.1 Member Function Documentation

### 5.25.1.1 draw()

```
void FlywheelPage::draw (
    vex::brain::lcd & screen,
    bool ,
    unsigned int ) [inline], [override], [virtual]
```

See also

[Page::draw](#)

Reimplemented from [screen::Page](#).

### 5.25.1.2 update()

```
void FlywheelPage::update (
    bool ,
    int ,
    int ) [inline], [override], [virtual]
```

#### See also

[Page::update](#)

Reimplemented from [screen::Page](#).

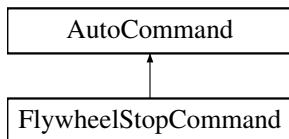
The documentation for this class was generated from the following file:

- [src/subsystems/flywheel.cpp](#)

## 5.26 FlywheelStopCommand Class Reference

```
#include <flywheel_commands.h>
```

Inheritance diagram for FlywheelStopCommand:



#### Public Member Functions

- [FlywheelStopCommand \(Flywheel &flywheel\)](#)
- bool [run \(\) override](#)

#### Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout \(\)](#)
- [AutoCommand \\* withTimeout \(double t\\_seconds\)](#)
- [AutoCommand \\* withCancelCondition \(Condition \\*true\\_to\\_end\)](#)

#### Additional Inherited Members

#### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition \\* true\\_to\\_end](#) = nullptr

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default\_timeout** = 10.0

### 5.26.1 Detailed Description

[AutoCommand](#) wrapper class for the stop function in the [Flywheel](#) class

### 5.26.2 Constructor & Destructor Documentation

#### 5.26.2.1 FlywheelStopCommand()

```
FlywheelStopCommand::FlywheelStopCommand (
    Flywheel & flywheel )
```

Construct a [FlywheelStopCommand](#)

##### Parameters

<i>flywheel</i>	the flywheel system we are commanding
-----------------	---------------------------------------

### 5.26.3 Member Function Documentation

#### 5.26.3.1 run()

```
bool FlywheelStopCommand::run ( ) [override], [virtual]
```

Run stop Overrides run from [AutoCommand](#)

##### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

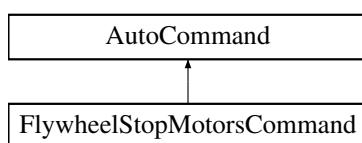
The documentation for this class was generated from the following files:

- include/utils/command\_structure/flywheel\_commands.h
- src/utils/command\_structure/flywheel\_commands.cpp

## 5.27 FlywheelStopMotorsCommand Class Reference

```
#include <flywheel_commands.h>
```

Inheritance diagram for FlywheelStopMotorsCommand:



## Public Member Functions

- `FlywheelStopMotorsCommand (Flywheel &flywheel)`
- `bool run () override`

## Public Member Functions inherited from [AutoCommand](#)

- `virtual void on_timeout ()`
- `AutoCommand * withTimeout (double t_seconds)`
- `AutoCommand * withCancelCondition (Condition *true_to_end)`

## Additional Inherited Members

## Public Attributes inherited from [AutoCommand](#)

- `double timeout_seconds = default_timeout`
- `Condition * true_to_end = nullptr`

## Static Public Attributes inherited from [AutoCommand](#)

- `static constexpr double default_timeout = 10.0`

### 5.27.1 Detailed Description

[AutoCommand](#) wrapper class for the stopMotors function in the [Flywheel](#) class

### 5.27.2 Constructor & Destructor Documentation

#### 5.27.2.1 FlywheelStopMotorsCommand()

```
FlywheelStopMotorsCommand::FlywheelStopMotorsCommand (
    Flywheel & flywheel )
```

Construct a FlywheelStopMotors Command

#### Parameters

<code>flywheel</code>	the flywheel system we are commanding
-----------------------	---------------------------------------

### 5.27.3 Member Function Documentation

#### 5.27.3.1 run()

```
bool FlywheelStopMotorsCommand::run ( ) [override], [virtual]
```

Run stop Overrides run from [AutoCommand](#)

**Returns**

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

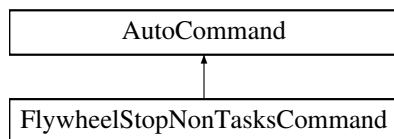
The documentation for this class was generated from the following files:

- include/utils/command\_structure/flywheel\_commands.h
- src/utils/command\_structure/flywheel\_commands.cpp

## 5.28 FlywheelStopNonTasksCommand Class Reference

```
#include <flywheel_commands.h>
```

Inheritance diagram for FlywheelStopNonTasksCommand:



### Additional Inherited Members

#### Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout \(\)](#)
- [AutoCommand \\* withTimeout](#) (double t\_seconds)
- [AutoCommand \\* withCancelCondition](#) ([Condition](#) \*true\_to\_end)

#### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

#### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

### 5.28.1 Detailed Description

[AutoCommand](#) wrapper class for the stopNonTasks function in the [Flywheel](#) class

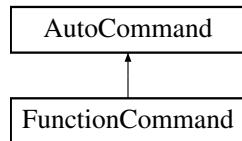
The documentation for this class was generated from the following files:

- include/utils/command\_structure/flywheel\_commands.h
- src/utils/command\_structure/flywheel\_commands.cpp

## 5.29 FunctionCommand Class Reference

```
#include <auto_command.h>
```

Inheritance diagram for FunctionCommand:



### Public Member Functions

- **FunctionCommand** (std::function< bool(void)> f)
- bool **run** ()

### Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)
- [AutoCommand](#) \* [withCancelCondition](#) ([Condition](#) \*true\_to\_end)

### Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

### 5.29.1 Detailed Description

[FunctionCommand](#) is fun and good way to do simple things Printing, launching nukes, and other quick and dirty one time things

## 5.29.2 Member Function Documentation

### 5.29.2.1 run()

```
bool FunctionCommand::run ( ) [inline], [virtual]
```

Executes the command Overridden by child classes

#### Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following file:

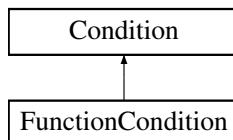
- include/utils/command\_structure/auto\_command.h

## 5.30 FunctionCondition Class Reference

[FunctionCondition](#) is a quick and dirty [Condition](#) to wrap some expression that should be evaluated at runtime.

```
#include <auto_command.h>
```

Inheritance diagram for FunctionCondition:



### Public Member Functions

- **FunctionCondition** (std::function< bool()> cond, std::function< void(void)> timeout=[ ]() {})
- bool [test](#) () override

### Public Member Functions inherited from [Condition](#)

- [Condition \\* Or](#) ([Condition](#) \*b)
- [Condition \\* And](#) ([Condition](#) \*b)

## 5.30.1 Detailed Description

[FunctionCondition](#) is a quick and dirty [Condition](#) to wrap some expression that should be evaluated at runtime.

## 5.30.2 Member Function Documentation

### 5.30.2.1 test()

```
bool FunctionCondition::test ( ) [override], [virtual]
```

Implements [Condition](#).

The documentation for this class was generated from the following files:

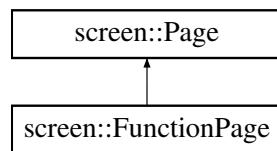
- include/utils/command\_structure/auto\_command.h
- src/utils/command\_structure/auto\_command.cpp

## 5.31 screen::FunctionPage Class Reference

Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything.

```
#include <screen.h>
```

Inheritance diagram for screen::FunctionPage:



### Public Member Functions

- [FunctionPage](#) (update\_func\_t update\_f, draw\_func\_t draw\_t)  
*Creates a function page.*
- void [update](#) (bool was\_pressed, int x, int y) override  
*update uses the supplied update function to update this page*
- void [draw](#) (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number) override  
*draw uses the supplied draw function to draw to the screen*

### 5.31.1 Detailed Description

Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything.

## 5.31.2 Constructor & Destructor Documentation

### 5.31.2.1 FunctionPage()

```
screen::FunctionPage::FunctionPage (
    update_func_t update_f,
    draw_func_t draw_f )
```

Creates a function page.

[FunctionPage](#).

**Parameters**

<i>update_f</i>	the function called every tick to respond to user input or do data collection
<i>draw_t</i>	the function called to draw to the screen
<i>update_f</i>	drawing function
<i>draw_f</i>	drawing function

**5.31.3 Member Function Documentation****5.31.3.1 draw()**

```
void screen::FunctionPage::draw (
    vex::brain::lcd & screen,
    bool first_draw,
    unsigned int frame_number ) [override], [virtual]
```

draw uses the supplied draw function to draw to the screen

**See also**

[Page::draw](#)

Reimplemented from [screen::Page](#).

**5.31.3.2 update()**

```
void screen::FunctionPage::update (
    bool was_pressed,
    int x,
    int y ) [override], [virtual]
```

update uses the supplied update function to update this page

**See also**

[Page::update](#)

Reimplemented from [screen::Page](#).

The documentation for this class was generated from the following files:

- include/subsystems/screen.h
- src/subsystems/screen.cpp

**5.32 GenericAuto Class Reference**

```
#include <generic_auto.h>
```

## Public Member Functions

- bool `run` (bool blocking)
- void `add` (state\_ptr new\_state)
- void `add_async` (state\_ptr async\_state)
- void `add_delay` (int ms)

### 5.32.1 Detailed Description

`GenericAuto` provides a pleasant interface for organizing an auto path steps of the path can be added with `add()` and when ready, calling `run()` will begin executing the path

### 5.32.2 Member Function Documentation

#### 5.32.2.1 add()

```
void GenericAuto::add (
    state_ptr new_state )
```

Add a new state to the autonomous via function point of type "bool (ptr\*)()"

Parameters

<code>new_state</code>	the function to run
------------------------	---------------------

#### 5.32.2.2 add\_async()

```
void GenericAuto::add_async (
    state_ptr async_state )
```

Add a new state to the autonomous via function point of type "bool (ptr\*)()" that will run asynchronously

Parameters

<code>async_state</code>	the function to run
--------------------------	---------------------

#### 5.32.2.3 add\_delay()

```
void GenericAuto::add_delay (
    int ms )
```

`add_delay` adds a period where the auto system will simply wait for the specified time

Parameters

<code>ms</code>	how long to wait in milliseconds
-----------------	----------------------------------

### 5.32.2.4 run()

```
bool GenericAuto::run (
    bool blocking )
```

The method that runs the autonomous. If 'blocking' is true, then this method will run through every state until it finished.

If blocking is false, then assuming every state is also non-blocking, the method will run through the current state in the list and return immediately.

#### Parameters

<i>blocking</i>	Whether or not to block the thread until all states have run
-----------------	--

#### Returns

true after all states have finished.

The documentation for this class was generated from the following files:

- include/utils/generic\_auto.h
- src/utils/generic\_auto.cpp

## 5.33 GraphDrawer Class Reference

### Public Member Functions

- [GraphDrawer](#) (int num\_samples, double lower\_bound, double upper\_bound, std::vector< vex::color > colors, size\_t num\_series=1)
   
*Creates a graph drawer with the specified number of series (each series is a separate line)*
- void [add\\_samples](#) (std::vector< point\_t > sample)
- void [add\\_samples](#) (std::vector< double > sample)
- void [draw](#) (vex::brain::lcd &screen, int x, int y, int width, int height)

### 5.33.1 Constructor & Destructor Documentation

#### 5.33.1.1 GraphDrawer()

```
GraphDrawer::GraphDrawer (
    int num_samples,
    double lower_bound,
    double upper_bound,
    std::vector< vex::color > colors,
    size_t num_series = 1 )
```

Creates a graph drawer with the specified number of series (each series is a separate line)

**Parameters**

<i>num_samples</i>	the number of samples to graph at a time (40 will graph the last 40 data points)
<i>lower_bound</i>	the bottom of the window when displaying (if <i>upper_bound</i> = <i>lower_bound</i> , auto calculate bounds)
<i>upper_bound</i>	the top of the window when displaying (if <i>upper_bound</i> = <i>lower_bound</i> , auto calculate bounds)
<i>colors</i>	the colors of the series. must be of size <i>num_series</i>
<i>num_series</i>	the number of series to graph

**5.33.2 Member Function Documentation****5.33.2.1 add\_samples() [1/2]**

```
void GraphDrawer::add_samples (
    std::vector< double > sample )
```

*add\_samples* adds a point to the graph, removing one from the back

**Parameters**

<i>sample</i>	a y coordinate of the next point to graph, the x coordinate is gotten from vex::timer::system(); (time in ms)
---------------	---

**5.33.2.2 add\_samples() [2/2]**

```
void GraphDrawer::add_samples (
    std::vector< point_t > new_samples )
```

*add\_samples* adds a point to the graph, removing one from the back

**Parameters**

<i>sample</i>	an x, y coordinate of the next point to graph
---------------	---

**5.33.2.3 draw()**

```
void GraphDrawer::draw (
    vex::brain::lcd & screen,
    int x,
    int y,
    int width,
    int height )
```

draws the graph to the screen in the constructor

**Parameters**

<i>x</i>	x position of the top left of the graphed region
----------	--

**Parameters**

<i>y</i>	y position of the top left of the graphed region
<i>width</i>	the width of the graphed region
<i>height</i>	the height of the graphed region

The documentation for this class was generated from the following files:

- include/utils/graph\_drawer.h
- src/utils/graph\_drawer.cpp

## 5.34 PurePursuit::hermite\_point Struct Reference

```
#include <pure_pursuit.h>
```

**Public Member Functions**

- `point_t getPoint () const`
- `Vector2D getTangent () const`

**Public Attributes**

- double `x`
- double `y`
- double `dir`
- double `mag`

### 5.34.1 Detailed Description

a position along the hermite path contains a position and orientation information that the robot would be at at this point

The documentation for this struct was generated from the following file:

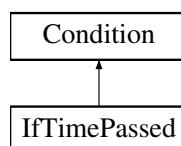
- include/utils/pure\_pursuit.h

## 5.35 IfTimePassed Class Reference

`IfTimePassed` tests based on time since the command controller was constructed. Returns true if elapsed time > time\_s.

```
#include <auto_command.h>
```

Inheritance diagram for IfTimePassed:



## Public Member Functions

- **IfTimePassed** (double time\_s)
- bool **test** () override

## Public Member Functions inherited from [Condition](#)

- [Condition \\* Or](#) ([Condition \\*b](#))
- [Condition \\* And](#) ([Condition \\*b](#))

### 5.35.1 Detailed Description

[IfTimePassed](#) tests based on time since the command controller was constructed. Returns true if elapsed time > time\_s.

### 5.35.2 Member Function Documentation

#### 5.35.2.1 test()

```
bool IfTimePassed::test ( ) [override], [virtual]
```

Implements [Condition](#).

The documentation for this class was generated from the following files:

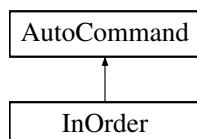
- include/utils/command\_structure/auto\_command.h
- src/utils/command\_structure/auto\_command.cpp

## 5.36 InOrder Class Reference

[InOrder](#) runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timeouts?

```
#include <auto_command.h>
```

Inheritance diagram for InOrder:



## Public Member Functions

- **InOrder** (const [InOrder](#) &other)=default
- **InOrder** (std::queue<[AutoCommand](#) \*> cmdqs)
- **InOrder** (std::initializer\_list<[AutoCommand](#) \*> cmdqs)
- bool **run** () override
- void **on\_timeout** () override

## Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand \\* withTimeout](#) (double t\_seconds)
- [AutoCommand \\* withCancelCondition](#) ([Condition](#) \*true\_to\_end)

## Additional Inherited Members

## Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition](#) \* [true\\_to\\_end](#) = nullptr

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

### 5.36.1 Detailed Description

[InOrder](#) runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timeouts?

[InOrder](#) runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timeouts?

### 5.36.2 Member Function Documentation

#### 5.36.2.1 [on\\_timeout\(\)](#)

```
void InOrder::on_timeout ( ) [override], [virtual]
```

What to do if we timeout instead of finishing. timeout is specified by the timeout seconds in the constructor

Reimplemented from [AutoCommand](#).

#### 5.36.2.2 [run\(\)](#)

```
bool InOrder::run ( ) [override], [virtual]
```

Executes the command Overridden by child classes

Returns

    true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- [include/utils/command\\_structure/auto\\_command.h](#)
- [src/utils/command\\_structure/auto\\_command.cpp](#)

## 5.37 Lift< T > Class Template Reference

```
#include <lift.h>
```

### Classes

- struct [lift\\_cfg\\_t](#)

### Public Member Functions

- [Lift](#) (motor\_group &lift\_motors, [lift\\_cfg\\_t](#) &lift\_cfg, map< T, double > &setpoint\_map, limit \*homing\_switch=NULL)
- void [control\\_continuous](#) (bool up\_ctrl, bool down\_ctrl)
- void [control\\_manual](#) (bool up\_btn, bool down\_btn, int volt\_up, int volt\_down)
- void [control\\_setpoints](#) (bool up\_step, bool down\_step, vector< T > pos\_list)
- bool [set\\_position](#) (T pos)
- bool [set\\_setpoint](#) (double val)
- double [get\\_setpoint](#) ()
- void [hold](#) ()
- void [home](#) ()
- bool [get\\_async](#) ()
- void [set\\_async](#) (bool val)
- void [set\\_sensor\\_function](#) (double(\*fn\_ptr)(void))
- void [set\\_sensor\\_reset](#) (void(\*fn\_ptr)(void))

### 5.37.1 Detailed Description

```
template<typename T>
class Lift< T >
```

LIFT A general class for lifts (e.g. 4bar, dr4bar, linear, etc) Uses a [PID](#) to hold the lift at a certain height under load, and to move the lift to different heights

#### Author

Ryan McGee

### 5.37.2 Constructor & Destructor Documentation

#### 5.37.2.1 Lift()

```
template<typename T >
Lift< T >::Lift (
    motor_group & lift_motors,
    lift_cfg_t & lift_cfg,
    map< T, double > & setpoint_map,
    limit * homing_switch = NULL ) [inline]
```

Construct the [Lift](#) object and begin the background task that controls the lift.

Usage example: /code{.cpp} enum Positions {UP, MID, DOWN}; map<Positions, double> setpt\_map { {DOWN, 0.0}, {MID, 0.5}, {UP, 1.0} }; Lift<Positions> my\_lift(motors, lift\_cfg, setpt\_map); /endcode

**Parameters**

<i>lift_motors</i>	A set of motors, all set that positive rotation correlates with the lift going up
<i>lift_cfg</i>	Lift characterization information; PID tunings and movement speeds
<i>setpoint_map</i>	A map of enum type T, in which each enum entry corresponds to a different lift height

**5.37.3 Member Function Documentation****5.37.3.1 control\_continuous()**

```
template<typename T >
void Lift< T >::control_continuous (
    bool up_ctrl,
    bool down_ctrl ) [inline]
```

Control the lift with an "up" button and a "down" button. Use PID to hold the lift when letting go.

**Parameters**

<i>up_ctrl</i>	Button controlling the "UP" motion
<i>down_ctrl</i>	Button controlling the "DOWN" motion

**5.37.3.2 control\_manual()**

```
template<typename T >
void Lift< T >::control_manual (
    bool up_btn,
    bool down_btn,
    int volt_up,
    int volt_down ) [inline]
```

Control the lift with manual controls (no holding voltage)

**Parameters**

<i>up_btn</i>	Raise the lift when true
<i>down_btn</i>	Lower the lift when true
<i>volt_up</i>	Motor voltage when raising the lift
<i>volt_down</i>	Motor voltage when lowering the lift

**5.37.3.3 control\_setpoints()**

```
template<typename T >
void Lift< T >::control_setpoints (
    bool up_step,
    bool down_step,
    vector< T > pos_list ) [inline]
```

Control the lift in "steps". When the "up" button is pressed, the lift will go to the next position as defined by pos\_list. Order matters!

#### Parameters

<i>up_step</i>	A button that increments the position of the lift.
<i>down_step</i>	A button that decrements the position of the lift.
<i>pos_list</i>	A list of positions for the lift to go through. The higher the index, the higher the lift should be (generally).

#### 5.37.3.4 get\_async()

```
template<typename T >
bool Lift< T >::get_async ( ) [inline]
```

#### Returns

whether or not the background thread is running the lift

#### 5.37.3.5 get\_setpoint()

```
template<typename T >
double Lift< T >::get_setpoint ( ) [inline]
```

#### Returns

The current setpoint for the lift

#### 5.37.3.6 hold()

```
template<typename T >
void Lift< T >::hold ( ) [inline]
```

Target the class's setpoint. Calculate the [PID](#) output and set the lift motors accordingly.

#### 5.37.3.7 home()

```
template<typename T >
void Lift< T >::home ( ) [inline]
```

A blocking function that automatically homes the lift based on a sensor or hard stop, and sets the position to 0. A watchdog times out after 3 seconds, to avoid damage.

#### 5.37.3.8 set\_async()

```
template<typename T >
void Lift< T >::set_async (
    bool val ) [inline]
```

Enables or disables the background task. Note that running the control functions, or set\_position functions will immediately re-enable the task for autonomous use.

**Parameters**

<i>val</i>	Whether or not the background thread should run the lift
------------	--

**5.37.3.9 set\_position()**

```
template<typename T >
bool Lift< T >::set_position (
    T pos ) [inline]
```

Enable the background task, and send the lift to a position, specified by the setpoint map from the constructor.

**Parameters**

<i>pos</i>	A lift position enum type
------------	---------------------------

**Returns**

True if the pid has reached the setpoint

**5.37.3.10 set\_sensor\_function()**

```
template<typename T >
void Lift< T >::set_sensor_function (
    double(*) (void) fn_ptr ) [inline]
```

Creates a custom hook for any other type of sensor to be used on the lift. Example: /code{.cpp} my\_lift.set\_sensor\_function( [](){return my\_sensor.position();} ); /endcode

**Parameters**

<i>fn_ptr</i>	Pointer to custom sensor function
---------------	-----------------------------------

**5.37.3.11 set\_sensor\_reset()**

```
template<typename T >
void Lift< T >::set_sensor_reset (
    void(*) (void) fn_ptr ) [inline]
```

Creates a custom hook to reset the sensor used in [set\\_sensor\\_function\(\)](#). Example: /code{.cpp} my\_lift.set\_sensor\_reset( my\_sensor.resetPosition ); /endcode

**5.37.3.12 set\_setpoint()**

```
template<typename T >
bool Lift< T >::set_setpoint (
    double val ) [inline]
```

Manually set a setpoint value for the lift [PID](#) to go to.

**Parameters**

<i>val</i>	<a href="#">Lift</a> setpoint, in motor revolutions or sensor units defined by <code>get_sensor</code> . Cannot be outside the softstops.
------------	---

**Returns**

True if the pid has reached the setpoint

The documentation for this class was generated from the following file:

- include/subsystems/lift.h

## 5.38 Lift< T >::lift\_cfg\_t Struct Reference

```
#include <lift.h>
```

**Public Attributes**

- double `up_speed`
- double `down_speed`
- double `softstop_up`
- double `softstop_down`
- [PID::pid\\_config\\_t lift\\_pid\\_cfg](#)

### 5.38.1 Detailed Description

```
template<typename T>
struct Lift< T >::lift_cfg_t
```

[lift\\_cfg\\_t](#) holds the physical parameter specifications of a lify system. includes:

- maximum speeds for the system
- softstops to stop the lift from hitting the hard stops too hard

The documentation for this struct was generated from the following file:

- include/subsystems/lift.h

## 5.39 Logger Class Reference

Class to simplify writing to files.

```
#include <logger.h>
```

## Public Member Functions

- `Logger (const std::string &filename)`  
*Create a logger that will save to a file.*
- `Logger (const Logger &) = delete`  
*copying not allowed*
- `Logger & operator= (const Logger &) = delete`  
*copying not allowed*
- `void Log (const std::string &s)`  
*Write a string to the log.*
- `void Log (LogLevel level, const std::string &s)`  
*Write a string to the log with a loglevel.*
- `void Logln (const std::string &s)`  
*Write a string and newline to the log.*
- `void Logln (LogLevel level, const std::string &s)`  
*Write a string and a newline to the log with a loglevel.*
- `void Logf (const char *fmt, ...)`  
*Write a formatted string to the log.*
- `void Logf (LogLevel level, const char *fmt, ...)`  
*Write a formatted string to the log with a loglevel.*

## Static Public Attributes

- `static constexpr int MAX_FORMAT_LEN = 512`  
*maximum size for a string to be before it's written*

### 5.39.1 Detailed Description

Class to simplify writing to files.

### 5.39.2 Constructor & Destructor Documentation

#### 5.39.2.1 Logger()

```
Logger::Logger (
    const std::string & filename ) [explicit]
```

Create a logger that will save to a file.

##### Parameters

<code>filename</code>	the file to save to
-----------------------	---------------------

### 5.39.3 Member Function Documentation

#### 5.39.3.1 Log() [1/2]

```
void Logger::Log (
    const std::string & s )
```

Write a string to the log.

##### Parameters

s	the string to write
---	---------------------

#### 5.39.3.2 Log() [2/2]

```
void Logger::Log (
    LogLevel level,
    const std::string & s )
```

Write a string to the log with a loglevel.

##### Parameters

level	the level to write. DEBUG, NOTICE, WARNING, ERROR, CRITICAL, TIME
s	the string to write

#### 5.39.3.3 Logf() [1/2]

```
void Logger::Logf (
    const char * fmt,
    ... )
```

Write a formatted string to the log.

##### Parameters

fmt	the format string (like printf)
...	the args

#### 5.39.3.4 Logf() [2/2]

```
void Logger::Logf (
    LogLevel level,
    const char * fmt,
    ... )
```

Write a formatted string to the log with a loglevel.

**Parameters**

<i>level</i>	the level to write. DEBUG, NOTICE, WARNING, ERROR, CRITICAL, TIME
<i>fmt</i>	the format string (like printf)
...	the args

**5.39.3.5 LogIn() [1/2]**

```
void Logger::LogIn (
    const std::string & s )
```

Write a string and newline to the log.

**Parameters**

<i>s</i>	the string to write
----------	---------------------

**5.39.3.6 LogIn() [2/2]**

```
void Logger::LogIn (
    LogLevel level,
    const std::string & s )
```

Write a string and a newline to the log with a loglevel.

**Parameters**

<i>level</i>	the level to write. DEBUG, NOTICE, WARNING, ERROR, CRITICAL, TIME
<i>s</i>	the string to write

The documentation for this class was generated from the following files:

- include/utils/logger.h
- src/utils/logger.cpp

**5.40 MotionController::m\_profile\_cfg\_t Struct Reference**

```
#include <motion_controller.h>
```

**Public Attributes**

- double **max\_v**  
*the maximum velocity the robot can drive*
- double **accel**  
*the most acceleration the robot can do*
- [PID::pid\\_config\\_t pid\\_cfg](#)  
*configuration parameters for the internal PID controller*
- [FeedForward::ff\\_config\\_t ff\\_cfg](#)  
*configuration parameters for the internal*

### 5.40.1 Detailed Description

`m_profile_config` holds all data the motion controller uses to plan paths. When motion profile is given a target to drive to, `max_v` and `accel` are used to make the trapezoid profile instructing the controller how to drive. `pid_cfg`, `ff_cfg` are used to find the motor outputs necessary to execute this path.

The documentation for this struct was generated from the following file:

- include/utils/controls/motion\_controller.h

## 5.41 Mat2 Struct Reference

### Public Member Functions

- `point_t operator* (const point_t p) const`

### Static Public Member Functions

- static `Mat2 FromRotationDegrees (double degrees)`

### Public Attributes

- double `X11`
- double `X12`
- double `X21`
- double `X22`

The documentation for this struct was generated from the following file:

- include/utils/geometry.h

## 5.42 MecanumDrive Class Reference

```
#include <mecanum_drive.h>
```

### Classes

- struct `mecanumdrive_config_t`

### Public Member Functions

- `MecanumDrive (vex::motor &left_front, vex::motor &right_front, vex::motor &left_rear, vex::motor &right_rear, vex::rotation *lateral_wheel=NULL, vex::inertial *imu=NULL, mecanumdrive_config_t *config=NULL)`
- void `drive_raw (double direction_deg, double magnitude, double rotation)`
- void `drive (double left_y, double left_x, double right_x, int power=2)`
- bool `auto_drive (double inches, double direction, double speed, bool gyro_correction=true)`
- bool `auto_turn (double degrees, double speed, bool ignore_imu=false)`

### 5.42.1 Detailed Description

A class representing the Mecanum drivetrain. Contains 4 motors, a possible IMU (intertial), and a possible undriven perpendicular wheel.

### 5.42.2 Constructor & Destructor Documentation

#### 5.42.2.1 MecanumDrive()

```
MecanumDrive::MecanumDrive (
    vex::motor & left_front,
    vex::motor & right_front,
    vex::motor & left_rear,
    vex::motor & right_rear,
    vex::rotation * lateral_wheel = NULL,
    vex::inertial * imu = NULL,
    mecanumdrive_config_t * config = NULL )
```

Create the Mecanum drivetrain object

### 5.42.3 Member Function Documentation

#### 5.42.3.1 auto\_drive()

```
bool MecanumDrive::auto_drive (
    double inches,
    double direction,
    double speed,
    bool gyro_correction = true )
```

Drive the robot in a straight line automatically. If the inertial was declared in the constructor, use it to correct while driving. If the lateral wheel was declared in the constructor, use it for more accurate positioning while strafing.

#### Parameters

<i>inches</i>	How far the robot should drive, in inches
<i>direction</i>	What direction the robot should travel in, in degrees. 0 is forward, +/-180 is reverse, clockwise is positive.
<i>speed</i>	The maximum speed the robot should travel, in percent: -1.0->+1.0
<i>gyro_correction</i>	=true Whether or not to use the gyro to help correct while driving. Will always be false if no gyro was declared in the constructor.

Drive the robot in a straight line automatically. If the inertial was declared in the constructor, use it to correct while driving. If the lateral wheel was declared in the constructor, use it for more accurate positioning while strafing.

#### Parameters

<i>inches</i>	How far the robot should drive, in inches
<i>direction</i>	What direction the robot should travel in, in degrees. 0 is forward, +/-180 is reverse, clockwise is positive.

**Parameters**

<i>speed</i>	The maximum speed the robot should travel, in percent: -1.0->+1.0
<i>gyro_correction</i>	= true Whether or not to use the gyro to help correct while driving. Will always be false if no gyro was declared in the constructor.

**Returns**

Whether or not the maneuver is complete.

**5.42.3.2 auto\_turn()**

```
bool MecanumDrive::auto_turn (
    double degrees,
    double speed,
    bool ignore_imu = false )
```

Autonomously turn the robot X degrees over it's center point. Uses a closed loop for control.

**Parameters**

<i>degrees</i>	How many degrees to rotate the robot. Clockwise positive.
<i>speed</i>	What percentage to run the motors at: 0.0 -> 1.0
<i>ignore_imu</i>	=false Whether or not to use the Inertial for determining angle. Will instead use circumference formula + robot's wheelbase + encoders to determine.

**Returns**

whether or not the robot has finished the maneuver

Autonomously turn the robot X degrees over it's center point. Uses a closed loop for control.

**Parameters**

<i>degrees</i>	How many degrees to rotate the robot. Clockwise positive.
<i>speed</i>	What percentage to run the motors at: 0.0 -> 1.0
<i>ignore_imu</i>	= false Whether or not to use the Inertial for determining angle. Will instead use circumference formula + robot's wheelbase + encoders to determine.

**Returns**

whether or not the robot has finished the maneuver

**5.42.3.3 drive()**

```
void MecanumDrive::drive (
    double left_y,
```

```
    double left_x,
    double right_x,
    int power = 2 )
```

Drive the robot with a mecanum-style / arcade drive. Inputs are in percent (-100.0 -> 100.0) straight from the controller. Controls are mixed, so the robot can drive forward / strafe / rotate all at the same time.

#### Parameters

<i>left_y</i>	left joystick, Y axis (forward / backwards)
<i>left_x</i>	left joystick, X axis (strafe left / right)
<i>right_x</i>	right joystick, X axis (rotation left / right)
<i>power</i>	=2 how much of a "curve" there should be on drive controls; better for low speed maneuvers. Leave blank for a default curve of 2 (higher means more fidelity)

Drive the robot with a mecanum-style / arcade drive. Inputs are in percent (-100.0 -> 100.0) straight from the controller. Controls are mixed, so the robot can drive forward / strafe / rotate all at the same time.

#### Parameters

<i>left_y</i>	left joystick, Y axis (forward / backwards)
<i>left_x</i>	left joystick, X axis (strafe left / right)
<i>right_x</i>	right joystick, X axis (rotation left / right)
<i>power</i>	=2 how much of a "curve" there should be on drive controls; better for low speed maneuvers. Leave blank for a default curve of 2 (higher means more fidelity)

### 5.42.3.4 `drive_raw()`

```
void MecanumDrive::drive_raw (
    double direction_deg,
    double magnitude,
    double rotation )
```

Drive the robot using vectors. This handles all the math required for mecanum control.

#### Parameters

<i>direction_deg</i>	the direction to drive the robot, in degrees. 0 is forward, 180 is back, clockwise is positive, counterclockwise is negative.
<i>magnitude</i>	How fast the robot should drive, in percent: 0.0->1.0
<i>rotation</i>	How fast the robot should rotate, in percent: -1.0->+1.0

The documentation for this class was generated from the following files:

- include/subsystems/mecanum\_drive.h
- src/subsystems/mecanum\_drive.cpp

## 5.43 MecanumDrive::mecanumdrive\_config\_t Struct Reference

```
#include <mecanum_drive.h>
```

### Public Attributes

- `PID::pid_config_t drive_pid_conf`
- `PID::pid_config_t drive_gyro_pid_conf`
- `PID::pid_config_t turn_pid_conf`
- `double drive_wheel_diam`
- `double lateral_wheel_diam`
- `double wheelbase_width`

### 5.43.1 Detailed Description

Configure the Mecanum drive [PID](#) tunings and robot configurations

The documentation for this struct was generated from the following file:

- `include/subsystems/mecanum_drive.h`

## 5.44 motion\_t Struct Reference

```
#include <trapezoid_profile.h>
```

### Public Attributes

- `double pos`  
*1d position at this point in time*
- `double vel`  
*1d velocity at this point in time*
- `double accel`  
*1d acceleration at this point in time*

### 5.44.1 Detailed Description

`motion_t` is a description of 1 dimensional motion at a point in time.

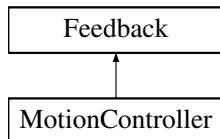
The documentation for this struct was generated from the following file:

- `include/utils/controls/trapezoid_profile.h`

## 5.45 MotionController Class Reference

```
#include <motion_controller.h>
```

Inheritance diagram for MotionController:



### Classes

- struct [m\\_profile\\_cfg\\_t](#)

### Public Member Functions

- [MotionController \(m\\_profile\\_cfg\\_t &config\)](#)  
*Construct a new Motion Controller object.*
- void [init \(double start\\_pt, double end\\_pt, double start\\_vel, double end\\_vel\) override](#)  
*Initialize the motion profile for a new movement This will also reset the PID and profile timers.*
- double [update \(double sensor\\_val\) override](#)  
*Update the motion profile with a new sensor value.*
- double [get \(\) override](#)
- void [set\\_limits \(double lower, double upper\) override](#)
- bool [is\\_on\\_target \(\) override](#)
- [motion\\_t get\\_motion \(\)](#)

### Static Public Member Functions

- static [FeedForward::ff\\_config\\_t tune\\_feedforward \(TankDrive &drive, OdometryTank &odometry, double pct=0.6, double duration=2\)](#)

### 5.45.1 Detailed Description

Motion Controller class

This class defines a top-level motion profile, which can act as an intermediate between a subsystem class and the motors themselves

This takes the constants kS, kV, kA, kP, kI, kD, max\_v and acceleration and wraps around a feedforward, [PID](#) and trapezoid profile. It does so with the following formula:

```
out = feedforward.calculate(motion_profile.get(time_s)) + pid.get(motion_profile.get(time_s))
```

For [PID](#) and Feedforward specific formulae, see [pid.h](#), [feedforward.h](#), and [trapezoid\\_profile.h](#)

#### Author

Ryan McGee

#### Date

7/13/2022

## 5.45.2 Constructor & Destructor Documentation

### 5.45.2.1 MotionController()

```
MotionController::MotionController (
    m_profile_cfg_t & config )
```

Construct a new Motion Controller object.

#### Parameters

<i>config</i>	The definition of how the robot is able to move max_v Maximum velocity the movement is capable of accel Acceleration / deceleration of the movement pid_cfg Definitions of kP, kI, and kD ff_cfg Definitions of kS, kV, and kA
---------------	--

## 5.45.3 Member Function Documentation

### 5.45.3.1 get()

```
double MotionController::get ( ) [override], [virtual]
```

#### Returns

the last saved result from the feedback controller

Implements [Feedback](#).

### 5.45.3.2 get\_motion()

```
motion_t MotionController::get_motion ( )
```

#### Returns

The current position, velocity and acceleration setpoints

### 5.45.3.3 init()

```
void MotionController::init (
    double start_pt,
    double end_pt,
    double start_vel,
    double end_vel ) [override], [virtual]
```

Initialize the motion profile for a new movement This will also reset the [PID](#) and profile timers.

#### Parameters

<i>start_pt</i>	Movement starting position
<i>end_pt</i>	Movement ending position
<i>start_vel</i>	Movement starting velocity
<i>end_vel</i>	Movement ending velocity

Implements [Feedback](#).

#### 5.45.3.4 is\_on\_target()

```
bool MotionController::is_on_target ( ) [override], [virtual]
```

##### Returns

Whether or not the movement has finished, and the [PID](#) confirms it is on target

Implements [Feedback](#).

#### 5.45.3.5 set\_limits()

```
void MotionController::set_limits (
    double lower,
    double upper ) [override], [virtual]
```

Clamp the upper and lower limits of the output. If both are 0, no limits should be applied. if limits are applied, the controller will not target any value below lower or above upper

##### Parameters

<i>lower</i>	upper limit
<i>upper</i>	lower limiet

Clamp the upper and lower limits of the output. If both are 0, no limits should be applied.

##### Parameters

<i>lower</i>	Upper limit
<i>upper</i>	Lower limit

Implements [Feedback](#).

#### 5.45.3.6 tune\_feedforward()

```
FeedForward::ff_config_t MotionController::tune_feedforward (
    TankDrive & drive,
    OdometryTank & odometry,
    double pct = 0.6,
    double duration = 2 ) [static]
```

This method attempts to characterize the robot's drivetrain and automatically tune the feedforward. It does this by first calculating the kS (voltage to overcome static friction) by slowly increasing the voltage until it moves.

Next is kV (voltage to sustain a certain velocity), where the robot will record it's steady-state velocity at 'pct' speed.

Finally, kA (voltage needed to accelerate by a certain rate), where the robot will record the entire movement's velocity and acceleration, record a plot of [X=(pct-kV\*V-kS), Y=(Acceleration)] along the movement, and since kA\*Accel = pct-kV\*V-kS, the reciprocal of the linear regression is the kA value.

**Parameters**

<i>drive</i>	The tankdrive to operate on
<i>odometry</i>	The robot's odometry subsystem
<i>pct</i>	Maximum velocity in percent (0->1.0)
<i>duration</i>	Amount of time the robot should be moving for the test

**Returns**

A tuned feedforward object

**5.45.3.7 update()**

```
double MotionController::update (
    double sensor_val ) [override], [virtual]
```

Update the motion profile with a new sensor value.

**Parameters**

<i>sensor_val</i>	Value from the sensor
-------------------	-----------------------

**Returns**

the motor input generated from the motion profile

Implements [Feedback](#).

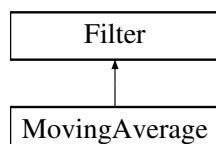
The documentation for this class was generated from the following files:

- include/utils/controls/motion\_controller.h
- src/utils/controls/motion\_controller.cpp

**5.46 MovingAverage Class Reference**

```
#include <moving_average.h>
```

Inheritance diagram for MovingAverage:



## Public Member Functions

- [MovingAverage \(int buffer\\_size\)](#)
- [MovingAverage \(int buffer\\_size, double starting\\_value\)](#)
- [void add\\_entry \(double n\)](#)
- [double get\\_value \(\) const](#)
- [int get\\_size \(\) const](#)

### 5.46.1 Detailed Description

#### [MovingAverage](#)

A moving average is a way of smoothing out noisy data. For many sensor readings, the noise is roughly symmetric around the actual value. This means that if you collect enough samples those that are too high are cancelled out by the samples that are too low leaving the real value.

The [MovingAverage](#) class provides a simple interface to do this smoothing from our noisy sensor values.

WARNING: because we need a lot of samples to get the actual value, the value given by the [MovingAverage](#) will 'lag' behind the actual value that the sensor is reading. Using a [MovingAverage](#) is thus a tradeoff between accuracy and lag time (more samples) vs. less accuracy and faster updating (less samples).

### 5.46.2 Constructor & Destructor Documentation

#### 5.46.2.1 [MovingAverage\(\)](#) [1/2]

```
MovingAverage::MovingAverage (
    int buffer_size )
```

Create a moving average calculator with 0 as the default value

##### Parameters

<i>buffer_size</i>	The size of the buffer. The number of samples that constitute a valid reading
--------------------	---

#### 5.46.2.2 [MovingAverage\(\)](#) [2/2]

```
MovingAverage::MovingAverage (
    int buffer_size,
    double starting_value )
```

Create a moving average calculator with a specified default value

##### Parameters

<i>buffer_size</i>	The size of the buffer. The number of samples that constitute a valid reading
<i>starting_value</i>	The value that the average will be before any data is added

### 5.46.3 Member Function Documentation

#### 5.46.3.1 add\_entry()

```
void MovingAverage::add_entry (
    double n ) [virtual]
```

Add a reading to the buffer Before: [ 1 1 2 2 3 3 ] => 2 ^ After: [ 2 1 2 2 3 3 ] => 2.16 ^

##### Parameters

<i>n</i>	the sample that will be added to the moving average.
----------	--

Implements [Filter](#).

#### 5.46.3.2 get\_size()

```
int MovingAverage::get_size ( ) const
```

How many samples the average is made from

##### Returns

the number of samples used to calculate this average

#### 5.46.3.3 get\_value()

```
double MovingAverage::get_value ( ) const [virtual]
```

Returns the average based off of all the samples collected so far

##### Returns

the calculated average. sum(samples)/numsamples

How many samples the average is made from

##### Returns

the number of samples used to calculate this average

Implements [Filter](#).

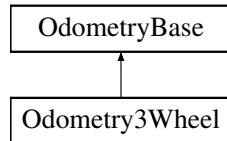
The documentation for this class was generated from the following files:

- include/utils/moving\_average.h
- src/utils/moving\_average.cpp

## 5.47 Odometry3Wheel Class Reference

```
#include <odometry_3wheel.h>
```

Inheritance diagram for Odometry3Wheel:



### Classes

- struct [odometry3wheel\\_cfg\\_t](#)

### Public Member Functions

- [Odometry3Wheel \(CustomEncoder &lside\\_fwd, CustomEncoder &rside\\_fwd, CustomEncoder &off\\_axis, odometry3wheel\\_cfg\\_t &cfg, bool is\\_async=true\)](#)
- [pose\\_t update \(\) override](#)
- [void tune \(vex::controller &con, TankDrive &drive\)](#)

### Public Member Functions inherited from [OdometryBase](#)

- [OdometryBase \(bool is\\_async\)](#)
- [pose\\_t get\\_position \(void\)](#)
- [virtual void set\\_position \(const pose\\_t &newpos=zero\\_pos\)](#)
- [void end\\_async \(\)](#)
- [double get\\_speed \(\)](#)
- [double get\\_accel \(\)](#)
- [double get\\_angular\\_speed\\_deg \(\)](#)
- [double get\\_angular\\_accel\\_deg \(\)](#)

### Additional Inherited Members

### Static Public Member Functions inherited from [OdometryBase](#)

- static int [background\\_task \(void \\*ptr\)](#)
- static double [pos\\_diff \(pose\\_t start\\_pos, pose\\_t end\\_pos\)](#)
- static double [rot\\_diff \(pose\\_t pos1, pose\\_t pos2\)](#)
- static double [smallest\\_angle \(double start\\_deg, double end\\_deg\)](#)

### Public Attributes inherited from [OdometryBase](#)

- bool [end\\_task = false](#)  
*end\_task is true if we instruct the odometry thread to shut down*

## Static Public Attributes inherited from [OdometryBase](#)

- static constexpr [pose\\_t zero\\_pos](#) = {.x=0.0L, .y=0.0L, .rot=90.0L}

## Protected Attributes inherited from [OdometryBase](#)

- vex::task \* [handle](#)
- vex::mutex [mut](#)
- [pose\\_t current\\_pos](#)
- double [speed](#)
- double [accel](#)
- double [ang\\_speed\\_deg](#)
- double [ang\\_accel\\_deg](#)

### 5.47.1 Detailed Description

#### [Odometry3Wheel](#)

This class handles the code for a standard 3-pod odometry setup, where there are 3 "pods" made up of undriven (dead) wheels connected to encoders in the following configuration:

Where O is the center of rotation. The robot will monitor the changes in rotation of these wheels and calculate the robot's X, Y and rotation on the field.

This is a "set and forget" class, meaning once the object is created, the robot will immediately begin tracking it's movement in the background.

#### Author

Ryan McGee

#### Date

Oct 31 2022

### 5.47.2 Constructor & Destructor Documentation

#### 5.47.2.1 [Odometry3Wheel\(\)](#)

```
Odometry3Wheel::Odometry3Wheel (
    CustomEncoder & lside_fwd,
    CustomEncoder & rside_fwd,
    CustomEncoder & off_axis,
    odometry3wheel_cfg_t & cfg,
    bool is_async = true )
```

Construct a new Odometry 3 Wheel object

**Parameters**

<i>lside_fwd</i>	left-side encoder reference
<i>rside_fwd</i>	right-side encoder reference
<i>off_axis</i>	off-axis (perpendicular) encoder reference
<i>cfg</i>	robot odometry configuration
<i>is_async</i>	true to constantly run in the background

**5.47.3 Member Function Documentation****5.47.3.1 tune()**

```
void Odometry3Wheel::tune (
    vex::controller & con,
    TankDrive & drive )
```

A guided tuning process to automatically find tuning parameters. This method is blocking, and returns when tuning has finished. Follow the instructions on the controller to complete the tuning process

**Parameters**

<i>con</i>	Controller reference, for screen and button control
<i>drive</i>	Drivetrain reference for robot control

A guided tuning process to automatically find tuning parameters. This method is blocking, and returns when tuning has finished. Follow the instructions on the controller to complete the tuning process

It is assumed the gear ratio and encoder PPR have been set correctly

**5.47.3.2 update()**

```
pose_t Odometry3Wheel::update ( ) [override], [virtual]
```

Update the current position of the robot once, using the current state of the encoders and the previous known location

**Returns**

the robot's updated position

Implements [OdometryBase](#).

The documentation for this class was generated from the following files:

- include/subsystems/odometry/odometry\_3wheel.h
- src/subsystems/odometry/odometry\_3wheel.cpp

## 5.48 Odometry3Wheel::odometry3wheel\_cfg\_t Struct Reference

```
#include <odometry_3wheel.h>
```

### Public Attributes

- double wheelbase\_dist
- double off\_axis\_center\_dist
- double wheel\_diam

### 5.48.1 Detailed Description

`odometry3wheel_cfg_t` holds all the specifications for how to calculate position with 3 encoders See the core wiki for what exactly each of these parameters measures

### 5.48.2 Member Data Documentation

#### 5.48.2.1 off\_axis\_center\_dist

```
double Odometry3Wheel::odometry3wheel_cfg_t::off_axis_center_dist
```

distance from the center of the robot to the center off axis wheel

#### 5.48.2.2 wheel\_diam

```
double Odometry3Wheel::odometry3wheel_cfg_t::wheel_diam
```

the diameter of the tracking wheel

#### 5.48.2.3 wheelbase\_dist

```
double Odometry3Wheel::odometry3wheel_cfg_t::wheelbase_dist
```

distance from the center of the left wheel to the center of the right wheel

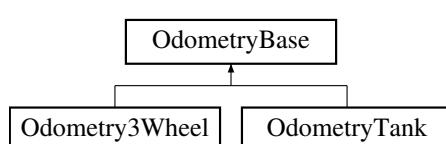
The documentation for this struct was generated from the following file:

- include/subsystems/odometry/odometry\_3wheel.h

## 5.49 OdometryBase Class Reference

```
#include <odometry_base.h>
```

Inheritance diagram for OdometryBase:



## Public Member Functions

- `OdometryBase (bool is_async)`
- `pose_t get_position ()`
- `virtual void set_position (const pose_t &newpos=zero_pos)`
- `virtual pose_t update ()=0`
- `void end_async ()`
- `double get_speed ()`
- `double get_accel ()`
- `double get_angular_speed_deg ()`
- `double get_angular_accel_deg ()`

## Static Public Member Functions

- `static int background_task (void *ptr)`
- `static double pos_diff (pose_t start_pos, pose_t end_pos)`
- `static double rot_diff (pose_t pos1, pose_t pos2)`
- `static double smallest_angle (double start_deg, double end_deg)`

## Public Attributes

- `bool end_task = false`  
*end\_task is true if we instruct the odometry thread to shut down*

## Static Public Attributes

- `static constexpr pose_t zero_pos = {.x=0.0L, .y=0.0L, .rot=90.0L}`

## Protected Attributes

- `vex::task * handle`
- `vex::mutex mut`
- `pose_t current_pos`
- `double speed`
- `double accel`
- `double ang_speed_deg`
- `double ang_accel_deg`

## 5.49.1 Detailed Description

### OdometryBase

This base class contains all the shared code between different implementations of odometry. It handles the asynchronous management, position input/output and basic math functions, and holds positional types specific to field orientation.

All future odometry implementations should extend this file and redefine `update()` function.

### Author

Ryan McGee

### Date

Aug 11 2021

## 5.49.2 Constructor & Destructor Documentation

### 5.49.2.1 OdometryBase()

```
OdometryBase::OdometryBase (
    bool is_async )
```

Construct a new Odometry Base object

#### Parameters

<i>is_async</i>	True to run constantly in the background, false to call <a href="#">update()</a> manually
-----------------	---

## 5.49.3 Member Function Documentation

### 5.49.3.1 background\_task()

```
int OdometryBase::background_task (
    void * ptr ) [static]
```

Function that runs in the background task. This function pointer is passed to the vex::task constructor.

#### Parameters

<i>ptr</i>	Pointer to <a href="#">OdometryBase</a> object
------------	--

#### Returns

Required integer return code. Unused.

### 5.49.3.2 end\_async()

```
void OdometryBase::end_async ( )
```

End the background task. Cannot be restarted. If the user wants to end the thread but keep the data up to date, they must run the [update\(\)](#) function manually from then on.

### 5.49.3.3 get\_accel()

```
double OdometryBase::get_accel ( )
```

Get the current acceleration

#### Returns

the acceleration rate of the robot (inch/s<sup>2</sup>)

#### 5.49.3.4 get\_angular\_accel\_deg()

```
double OdometryBase::get_angular_accel_deg ( )
```

Get the current angular acceleration in degrees

##### Returns

the angular acceleration at which we are turning (deg/s<sup>2</sup>)

#### 5.49.3.5 get\_angular\_speed\_deg()

```
double OdometryBase::get_angular_speed_deg ( )
```

Get the current angular speed in degrees

##### Returns

the angular velocity at which we are turning (deg/s)

#### 5.49.3.6 get\_position()

```
pose_t OdometryBase::get_position ( void )
```

Gets the current position and rotation

##### Returns

the position that the odometry believes the robot is at

Gets the current position and rotation

#### 5.49.3.7 get\_speed()

```
double OdometryBase::get_speed ( )
```

Get the current speed

##### Returns

the speed at which the robot is moving and grooving (inch/s)

#### 5.49.3.8 pos\_diff()

```
double OdometryBase::pos_diff ( pose_t start_pos, pose_t end_pos ) [static]
```

Get the distance between two points

**Parameters**

<i>start_pos</i>	distance from this point
<i>end_pos</i>	to this point

**Returns**

the euclidean distance between start\_pos and end\_pos

**5.49.3.9 rot\_diff()**

```
double OdometryBase::rot_diff (
    pose_t pos1,
    pose_t pos2 ) [static]
```

Get the change in rotation between two points

**Parameters**

<i>pos1</i>	position with initial rotation
<i>pos2</i>	position with final rotation

**Returns**

change in rotation between pos1 and pos2

Get the change in rotation between two points

**5.49.3.10 set\_position()**

```
void OdometryBase::set_position (
    const pose_t & newpos = zero_pos ) [virtual]
```

Sets the current position of the robot

**Parameters**

<i>newpos</i>	the new position that the odometry will believe it is at
---------------	--

Sets the current position of the robot

Reimplemented in [OdometryTank](#).

**5.49.3.11 smallest\_angle()**

```
double OdometryBase::smallest_angle (
    double start_deg,
    double end_deg ) [static]
```

Get the smallest difference in angle between a start heading and end heading. Returns the difference between -180 degrees and +180 degrees, representing the robot turning left or right, respectively.

#### Parameters

<code>start_deg</code>	initial angle (degrees)
<code>end_deg</code>	final angle (degrees)

#### Returns

the smallest angle from the initial to the final angle. This takes into account the wrapping of rotations around 360 degrees

Get the smallest difference in angle between a start heading and end heading. Returns the difference between -180 degrees and +180 degrees, representing the robot turning left or right, respectively.

### 5.49.3.12 `update()`

```
virtual pose_t OdometryBase::update ( ) [pure virtual]
```

Update the current position on the field based on the sensors

#### Returns

the location that the robot is at after the odometry does its calculations

Implemented in [Odometry3Wheel](#), and [OdometryTank](#).

## 5.49.4 Member Data Documentation

### 5.49.4.1 `accel`

```
double OdometryBase::accel [protected]
```

the rate at which we are accelerating (inch/s<sup>2</sup>)

### 5.49.4.2 `ang_accel_deg`

```
double OdometryBase::ang_accel_deg [protected]
```

the rate at which we are accelerating our turn (deg/s<sup>2</sup>)

### 5.49.4.3 `ang_speed_deg`

```
double OdometryBase::ang_speed_deg [protected]
```

the speed at which we are turning (deg/s)

#### 5.49.4.4 current\_pos

`pose_t OdometryBase::current_pos [protected]`

Current position of the robot in terms of x,y,rotation

#### 5.49.4.5 handle

`vex::task* OdometryBase::handle [protected]`

handle to the vex task that is running the odometry code

#### 5.49.4.6 mut

`vex::mutex OdometryBase::mut [protected]`

Mutex to control multithreading

#### 5.49.4.7 speed

`double OdometryBase::speed [protected]`

the speed at which we are travelling (inch/s)

#### 5.49.4.8 zero\_pos

`constexpr pose_t OdometryBase::zero_pos = { .x=0.0L, .y=0.0L, .rot=90.0L } [inline], [static], [constexpr]`

Zeroed position. X=0, Y=0, Rotation= 90 degrees

The documentation for this class was generated from the following files:

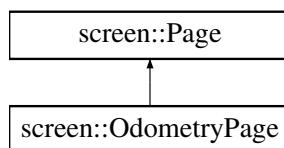
- include/subsystems/odometry/odometry\_base.h
- src/subsystems/odometry/odometry\_base.cpp

## 5.50 screen::OdometryPage Class Reference

a page that shows odometry position and rotation and a map (if an sd card with the file is on)

```
#include <screen.h>
```

Inheritance diagram for screen::OdometryPage:



## Public Member Functions

- [OdometryPage \(OdometryBase &odom, double robot\\_width, double robot\\_height, bool do\\_trail\)](#)  
*Create an odometry trail. Make sure odometry is initialized before now.*
- void [update \(bool was\\_pressed, int x, int y\) override](#)
- void [draw \(vex::brain::lcd &, bool first\\_draw, unsigned int frame\\_number\) override](#)

### 5.50.1 Detailed Description

a page that shows odometry position and rotation and a map (if an sd card with the file is on)

### 5.50.2 Constructor & Destructor Documentation

#### 5.50.2.1 OdometryPage()

```
screen::OdometryPage::OdometryPage (
    OdometryBase & odom,
    double robot_width,
    double robot_height,
    bool do_trail )
```

Create an odometry trail. Make sure odometry is initialized before now.

#### Parameters

<i>odom</i>	the odometry system to monitor
<i>robot_width</i>	the width (side to side) of the robot in inches. Used for visualization
<i>robot_height</i>	the robot_height (front to back) of the robot in inches. Used for visualization
<i>do_trail</i>	whether or not to calculate and draw the trail. Drawing and storing takes a very <i>slight</i> extra amount of processing power

### 5.50.3 Member Function Documentation

#### 5.50.3.1 draw()

```
void screen::OdometryPage::draw (
    vex::brain::lcd & scr,
    bool first_draw,
    unsigned int frame_number ) [override], [virtual]
```

#### See also

[Page::draw](#)

Reimplemented from [screen::Page](#).

### 5.50.3.2 update()

```
void screen::OdometryPage::update (
    bool was_pressed,
    int x,
    int y ) [override], [virtual]
```

#### See also

[Page::update](#)

Reimplemented from [screen::Page](#).

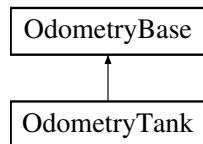
The documentation for this class was generated from the following files:

- include/subsystems/screen.h
- src/subsystems/screen.cpp

## 5.51 OdometryTank Class Reference

```
#include <odometry_tank.h>
```

Inheritance diagram for OdometryTank:



#### Public Member Functions

- [OdometryTank](#) (vex::motor\_group &left\_side, vex::motor\_group &right\_side, [robot\\_specs\\_t](#) &config, vex::inertial \*imu=NULL, bool is\_async=true)
- [OdometryTank](#) ([CustomEncoder](#) &left\_custom\_enc, [CustomEncoder](#) &right\_custom\_enc, [robot\\_specs\\_t](#) &config, vex::inertial \*imu=NULL, bool is\_async=true)
- [OdometryTank](#) (vex::encoder &left\_vex\_enc, vex::encoder &right\_vex\_enc, [robot\\_specs\\_t](#) &config, vex::inertial \*imu=NULL, bool is\_async=true)
- [pose\\_t update](#) () override
- void [set\\_position](#) (const [pose\\_t](#) &newpos=[zero\\_pos](#)) override

#### Public Member Functions inherited from [OdometryBase](#)

- [OdometryBase](#) (bool is\_async)
- [pose\\_t get\\_position](#) (void)
- void [end\\_async](#) ()
- double [get\\_speed](#) ()
- double [get\\_accel](#) ()
- double [get\\_angular\\_speed\\_deg](#) ()
- double [get\\_angular\\_accel\\_deg](#) ()

## Additional Inherited Members

### Static Public Member Functions inherited from [OdometryBase](#)

- static int [background\\_task](#) (void \*ptr)
- static double [pos\\_diff](#) ([pose\\_t](#) start\_pos, [pose\\_t](#) end\_pos)
- static double [rot\\_diff](#) ([pose\\_t](#) pos1, [pose\\_t](#) pos2)
- static double [smallest\\_angle](#) (double start\_deg, double end\_deg)

### Public Attributes inherited from [OdometryBase](#)

- bool [end\\_task](#) = false  
*end\_task is true if we instruct the odometry thread to shut down*

### Static Public Attributes inherited from [OdometryBase](#)

- static constexpr [pose\\_t](#) [zero\\_pos](#) = {.x=0.0L, .y=0.0L, .rot=90.0L}

### Protected Attributes inherited from [OdometryBase](#)

- vex::task \* [handle](#)
- vex::mutex [mut](#)
- [pose\\_t](#) [current\\_pos](#)
- double [speed](#)
- double [accel](#)
- double [ang\\_speed\\_deg](#)
- double [ang\\_accel\\_deg](#)

## 5.51.1 Detailed Description

[OdometryTank](#) defines an odometry system for a tank drivetrain. This requires encoders in the same orientation as the drive wheels. Odometry is a "start and forget" subsystem, which means once it's created and configured, it will constantly run in the background and track the robot's X, Y and rotation coordinates.

## 5.51.2 Constructor & Destructor Documentation

### 5.51.2.1 [OdometryTank\(\)](#) [1/3]

```
OdometryTank::OdometryTank (
    vex::motor_group & left_side,
    vex::motor_group & right_side,
    robot\_specs\_t & config,
    vex::inertial * imu = NULL,
    bool is_async = true )
```

Initialize the Odometry module, calculating position from the drive motors.

**Parameters**

<i>left_side</i>	The left motors
<i>right_side</i>	The right motors
<i>config</i>	the specifications that supply the odometry with descriptions of the robot. See <a href="#">robot_specs_t</a> for what is contained
<i>imu</i>	The robot's inertial sensor. If not included, rotation is calculated from the encoders.
<i>is_async</i>	If true, position will be updated in the background continuously. If false, the programmer will have to manually call <a href="#">update()</a> .

**5.51.2.2 OdometryTank() [2/3]**

```
OdometryTank::OdometryTank (
    CustomEncoder & left_custom_enc,
    CustomEncoder & right_custom_enc,
    robot_specs_t & config,
    vex::inertial * imu = NULL,
    bool is_async = true )
```

Initialize the Odometry module, calculating position from the drive motors.

**Parameters**

<i>left_custom_enc</i>	The left custom encoder
<i>right_custom_enc</i>	The right custom encoder
<i>config</i>	the specifications that supply the odometry with descriptions of the robot. See <a href="#">robot_specs_t</a> for what is contained
<i>imu</i>	The robot's inertial sensor. If not included, rotation is calculated from the encoders.
<i>is_async</i>	If true, position will be updated in the background continuously. If false, the programmer will have to manually call <a href="#">update()</a> .

**5.51.2.3 OdometryTank() [3/3]**

```
OdometryTank::OdometryTank (
    vex::encoder & left_vex_enc,
    vex::encoder & right_vex_enc,
    robot_specs_t & config,
    vex::inertial * imu = NULL,
    bool is_async = true )
```

Initialize the Odometry module, calculating position from the drive motors.

**Parameters**

<i>left_vex_enc</i>	The left vex encoder
<i>right_vex_enc</i>	The right vex encoder
<i>config</i>	the specifications that supply the odometry with descriptions of the robot. See <a href="#">robot_specs_t</a> for what is contained
<i>imu</i>	The robot's inertial sensor. If not included, rotation is calculated from the encoders.
<i>is_async</i>	If true, position will be updated in the background continuously. If false, the programmer will have to manually call <a href="#">update()</a> .

### 5.51.3 Member Function Documentation

#### 5.51.3.1 set\_position()

```
void OdometryTank::set_position (
    const pose_t & newpos = zero_pos ) [override], [virtual]
```

set\_position tells the odometry to place itself at a position

##### Parameters

<i>newpos</i>	the position the odometry will take
---------------	-------------------------------------

Resets the position and rotational data to the input.

Reimplemented from [OdometryBase](#).

#### 5.51.3.2 update()

```
pose_t OdometryTank::update ( ) [override], [virtual]
```

Update the current position on the field based on the sensors

##### Returns

the position that odometry has calculated itself to be at

Update, store and return the current position of the robot. Only use if not initializing with a separate thread.

Implements [OdometryBase](#).

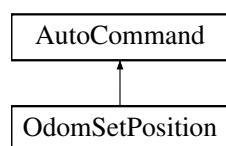
The documentation for this class was generated from the following files:

- include/subsystems/odometry/odometry\_tank.h
- src/subsystems/odometry/odometry\_tank.cpp

## 5.52 OdomSetPosition Class Reference

```
#include <drive_commands.h>
```

Inheritance diagram for OdomSetPosition:



## Public Member Functions

- `OdomSetPosition (OdometryBase &odom, const pose_t &newpos=OdometryBase::zero_pos)`
- `bool run () override`

## Public Member Functions inherited from [AutoCommand](#)

- `virtual void on_timeout ()`
- `AutoCommand * withTimeout (double t_seconds)`
- `AutoCommand * withCancelCondition (Condition *true_to_end)`

## Additional Inherited Members

## Public Attributes inherited from [AutoCommand](#)

- `double timeout_seconds = default_timeout`
- `Condition * true_to_end = nullptr`

## Static Public Attributes inherited from [AutoCommand](#)

- `static constexpr double default_timeout = 10.0`

### 5.52.1 Detailed Description

[AutoCommand](#) wrapper class for the set\_position function in the Odometry class

### 5.52.2 Constructor & Destructor Documentation

#### 5.52.2.1 OdomSetPosition()

```
OdomSetPosition::OdomSetPosition (
    OdometryBase & odom,
    const pose_t & newpos = OdometryBase::zero_pos )
```

constructs a new [OdomSetPosition](#) command

##### Parameters

<code>odom</code>	the odometry system we are setting
<code>newpos</code>	the position we are telling the odometry to take. defaults to (0, 0), angle = 90

Construct an Odometry set pos

##### Parameters

<code>odom</code>	the odometry system we are setting
<code>newpos</code>	the now position to set the odometry to

### 5.52.3 Member Function Documentation

#### 5.52.3.1 run()

```
bool OdomSetPosition::run ( ) [override], [virtual]
```

Run set\_position Overrides run from [AutoCommand](#)

##### Returns

true when execution is complete, false otherwise

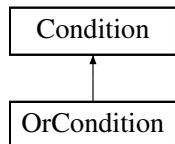
Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command\_structure/drive\_commands.h
- src/utils/command\_structure/drive\_commands.cpp

## 5.53 OrCondition Class Reference

Inheritance diagram for OrCondition:



### Public Member Functions

- [OrCondition \(Condition \\*A, Condition \\*B\)](#)
- bool [test \(\) override](#)

### Public Member Functions inherited from [Condition](#)

- [Condition \\* Or \(Condition \\*b\)](#)
- [Condition \\* And \(Condition \\*b\)](#)

### 5.53.1 Member Function Documentation

#### 5.53.1.1 test()

```
bool OrCondition::test ( ) [inline], [override], [virtual]
```

Implements [Condition](#).

The documentation for this class was generated from the following file:

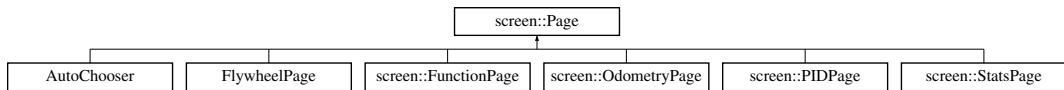
- src/utils/command\_structure/auto\_command.cpp

## 5.54 screen::Page Class Reference

[Page](#) describes one part of the screen slideshow.

```
#include <screen.h>
```

Inheritance diagram for screen::Page:



### Public Member Functions

- virtual void [update](#) (bool *was\_pressed*, int *x*, int *y*)  
*collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this [Page](#) (only drawn page gets touch updates))*
- virtual void [draw](#) (vex::brain::lcd &*screen*, bool *first\_draw*, unsigned int *frame\_number*)  
*draw stored data to the screen (runs at 10 hz and only runs if this page is in front)*

### 5.54.1 Detailed Description

[Page](#) describes one part of the screen slideshow.

### 5.54.2 Member Function Documentation

#### 5.54.2.1 draw()

```
virtual void screen::Page::draw (
    vex::brain::lcd & screen,
    bool first_draw,
    unsigned int frame_number ) [virtual]
```

draw stored data to the screen (runs at 10 hz and only runs if this page is in front)

##### Parameters

<i>first_draw</i>	true if we just switched to this page
<i>frame_number</i>	frame of drawing we are on (basically an animation tick)

Reimplemented in [AutoChooser](#), [screen::StatsPage](#), [screen::OdometryPage](#), [screen::FunctionPage](#), [screen::PIDPage](#), and [FlywheelPage](#).

#### 5.54.2.2 update()

```
virtual void screen::Page::update (
    bool was_pressed,
```

```
int x,
int y ) [virtual]
```

collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this [Page](#) (only drawn page gets touch updates))

#### Parameters

<code>was_pressed</code>	true if the screen has been pressed
<code>x</code>	x position of screen press (if the screen was pressed)
<code>y</code>	y position of screen press (if the screen was pressed)

Reimplemented in [AutoChooser](#), [screen::StatsPage](#), [screen::OdometryPage](#), [screen::FunctionPage](#), [screen::PIDPage](#), and [FlywheelPage](#).

The documentation for this class was generated from the following file:

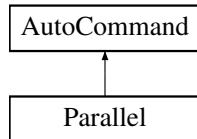
- include/subsystems/screen.h

## 5.55 Parallel Class Reference

[Parallel](#) runs multiple commands in parallel and waits for all to finish before continuing. if none finish before this command's timeout, it will call `on_timeout` on all children continue.

```
#include <auto_command.h>
```

Inheritance diagram for Parallel:



### Public Member Functions

- `Parallel (std::initializer_list< AutoCommand * > cmd)`
- `bool run () override`
- `void on_timeout () override`

### Public Member Functions inherited from [AutoCommand](#)

- `AutoCommand * withTimeout (double t_seconds)`
- `AutoCommand * withCancelCondition (Condition *true_to_end)`

### Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- `double timeout_seconds = default_timeout`
- `Condition * true_to_end = nullptr`

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default\_timeout** = 10.0

### 5.55.1 Detailed Description

[Parallel](#) runs multiple commands in parallel and waits for all to finish before continuing. if none finish before this command's timeout, it will call `on_timeout` on all children continue.

### 5.55.2 Member Function Documentation

#### 5.55.2.1 `on_timeout()`

```
void Parallel::on_timeout ( ) [override], [virtual]
```

What to do if we timeout instead of finishing. timeout is specified by the timeout seconds in the constructor

Reimplemented from [AutoCommand](#).

#### 5.55.2.2 `run()`

```
bool Parallel::run ( ) [override], [virtual]
```

Executes the command Overridden by child classes

##### Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command\_structure/auto\_command.h
- src/utils/command\_structure/auto\_command.cpp

## 5.56 `parallel_runner_info` Struct Reference

### Public Attributes

- int **index**
- std::vector< vex::task \* > \* **runners**
- [AutoCommand](#) \* **cmd**

The documentation for this struct was generated from the following file:

- src/utils/command\_structure/auto\_command.cpp

## 5.57 PurePursuit::Path Class Reference

```
#include <pure_pursuit.h>
```

### Public Member Functions

- [Path \(std::vector< point\\_t > points, double radius\)](#)
- [std::vector< point\\_t > get\\_points \(\)](#)
- [double get\\_radius \(\)](#)
- [bool is\\_valid \(\)](#)

### 5.57.1 Detailed Description

Wrapper for a vector of points, checking if any of the points are too close for pure pursuit

### 5.57.2 Constructor & Destructor Documentation

#### 5.57.2.1 Path()

```
PurePursuit::Path::Path (
    std::vector< point_t > points,
    double radius )
```

Create a [Path](#)

#### Parameters

<i>points</i>	the points that make up the path
<i>radius</i>	the lookahead radius for pure pursuit

### 5.57.3 Member Function Documentation

#### 5.57.3.1 get\_points()

```
std::vector< point_t > PurePursuit::Path::get_points ( )
```

Get the points associated with this [Path](#)

#### 5.57.3.2 get\_radius()

```
double PurePursuit::Path::get_radius ( )
```

Get the radius associated with this [Path](#)

### 5.57.3.3 `is_valid()`

```
bool PurePursuit::Path::is_valid ( )
```

Get whether this path will behave as expected

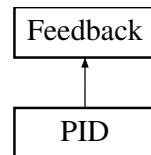
The documentation for this class was generated from the following files:

- include/utils/pure\_pursuit.h
- src/utils/pure\_pursuit.cpp

## 5.58 PID Class Reference

```
#include <pid.h>
```

Inheritance diagram for PID:



### Classes

- struct `pid_config_t`

### Public Types

- enum `ERROR_TYPE` { `LINEAR` , `ANGULAR` }

### Public Member Functions

- `PID (pid_config_t &config)`
- void `init (double start_pt, double set_pt, double start_vel=0, double end_vel=0)` override
- double `update (double sensor_val)` override
- double `get_sensor_val ()`  
*gets the sensor value that we were last updated with*
- double `get ()` override
- void `set_limits (double lower, double upper)` override
- bool `is_on_target ()` override
- void `reset ()`
- double `get_error ()`
- double `get_target ()`
- void `set_target (double target)`

## Public Attributes

- `pid_config_t & config`

*configuration struct for this controller. see `pid_config_t` for information about what this contains*

## 5.58.1 Detailed Description

### PID Class

Defines a standard feedback loop using the constants kP, kI, kD, deadband, and on\_target\_time. The formula is:

$$\text{out} = \text{kP} * \text{error} + \text{kI} * \text{integral}(\text{d Error}) + \text{kD} * (\text{dError}/\text{dt})$$

The `PID` object will determine it is "on target" when the error is within the deadband, for a duration of on\_target\_time

### Author

Ryan McGee

### Date

4/3/2020

## 5.58.2 Member Enumeration Documentation

### 5.58.2.1 ERROR\_TYPE

enum `PID::ERROR_TYPE`

An enum to distinguish between a linear and angular calculation of `PID` error.

## 5.58.3 Constructor & Destructor Documentation

### 5.58.3.1 PID()

```
PID::PID (
    pid_config_t & config )
```

Create the `PID` object

#### Parameters

<code>config</code>	the configuration data for this controller
---------------------	--

Create the `PID` object

## 5.58.4 Member Function Documentation

### 5.58.4.1 get()

```
double PID::get () [override], [virtual]
```

Gets the current [PID](#) out value, from when [update\(\)](#) was last run

#### Returns

the Out value of the controller (voltage, RPM, whatever the [PID](#) controller is controlling)

Gets the current [PID](#) out value, from when [update\(\)](#) was last run

Implements [Feedback](#).

### 5.58.4.2 get\_error()

```
double PID::get_error ()
```

Get the delta between the current sensor data and the target

#### Returns

the error calculated. how it is calculated depends on error\_method specified in [pid\\_config\\_t](#)

Get the delta between the current sensor data and the target

### 5.58.4.3 get\_sensor\_val()

```
double PID::get_sensor_val ()
```

gets the sensor value that we were last updated with

#### Returns

`sensor_val`

### 5.58.4.4 get\_target()

```
double PID::get_target ()
```

Get the [PID](#)'s target

#### Returns

the target the [PID](#) controller is trying to achieve

### 5.58.4.5 init()

```
void PID::init (
    double start_pt,
    double set_pt,
    double start_vel = 0,
    double end_vel = 0 ) [override], [virtual]
```

Inherited from [Feedback](#) for interoperability. Update the setpoint and reset integral accumulation

`start_pt` can be safely ignored in this feedback controller

**Parameters**

<i>start_pt</i>	completely ignored for <b>PID</b> . necessary to satisfy <a href="#">Feedback</a> base
<i>set_pt</i>	sets the target of the <b>PID</b> controller
<i>start_vel</i>	completely ignored for <b>PID</b> . necessary to satisfy <a href="#">Feedback</a> base
<i>end_vel</i>	sets the target end velocity of the <b>PID</b> controller

Implements [Feedback](#).

**5.58.4.6 is\_on\_target()**

```
bool PID::is_on_target ( ) [override], [virtual]
```

Checks if the **PID** controller is on target.

**Returns**

true if the loop is within [deadband] for [on\_target\_time] seconds

Returns true if the loop is within [deadband] for [on\_target\_time] seconds

Implements [Feedback](#).

**5.58.4.7 reset()**

```
void PID::reset ( )
```

Reset the **PID** loop by resetting time since 0 and accumulated error.

**5.58.4.8 set\_limits()**

```
void PID::set_limits (
    double lower,
    double upper ) [override], [virtual]
```

Set the limits on the **PID** out. The **PID** out will "clip" itself to be between the limits.

**Parameters**

<i>lower</i>	the lower limit. the <b>PID</b> controller will never command the output go below <i>lower</i>
<i>upper</i>	the upper limit. the <b>PID</b> controller will never command the output go higher than <i>upper</i>

Set the limits on the **PID** out. The **PID** out will "clip" itself to be between the limits.

Implements [Feedback](#).

#### 5.58.4.9 set\_target()

```
void PID::set_target (
    double target )
```

Set the target for the [PID](#) loop, where the robot is trying to end up

##### Parameters

<i>target</i>	the sensor reading we would like to achieve
---------------	---

Set the target for the [PID](#) loop, where the robot is trying to end up

#### 5.58.4.10 update()

```
double PID::update (
    double sensor_val ) [override], [virtual]
```

Update the [PID](#) loop by taking the time difference from last update, and running the [PID](#) formula with the new sensor data

##### Parameters

<i>sensor_val</i>	the distance, angle, encoder position or whatever it is we are measuring
-------------------	--

##### Returns

the new output. What would be returned by [PID::get\(\)](#)

Implements [Feedback](#).

The documentation for this class was generated from the following files:

- include/utils/controls/pid.h
- src/utils/controls/pid.cpp

## 5.59 PID::pid\_config\_t Struct Reference

```
#include <pid.h>
```

### Public Attributes

- double **p**  
*proportional coefficient p \* error()*
- double **i**  
*integral coefficient i \* integral(error)*
- double **d**

- **double deadband**  
*derivative coefficient d \* derivative(error)*
- **double on\_target\_time**  
*at what threshold are we close enough to be finished*
- **ERROR\_TYPE error\_method**  
*the time in seconds that we have to be on target for to say we are officially at the target*
- **Linear or angular.** whether to do error as a simple subtraction or to wrap.

### 5.59.1 Detailed Description

`pid_config_t` holds the configuration parameters for a pid controller. In addition to the constant of proportional, integral and derivative, these parameters include:

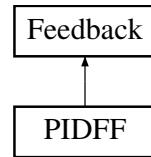
- **deadband** -
- **on\_target\_time** - for how long do we have to be at the target to stop. As well, `pid_config_t` holds an error type which determines whether errors should be calculated as if the sensor position is a measure of distance or an angle

The documentation for this struct was generated from the following file:

- `include/utils/controls/pid.h`

## 5.60 PIDFF Class Reference

Inheritance diagram for PIDFF:



### Public Member Functions

- **PIDFF (PID::pid\_config\_t &pid\_cfg, FeedForward::ff\_config\_t &ff\_cfg)**
- void **init** (double start\_pt, double set\_pt, double start\_vel, double end\_vel) override
- void **set\_target** (double set\_pt)
- double **update** (double val) override
- double **update** (double val, double vel\_setpt, double a\_setpt=0)
- double **get** () override
- void **set\_limits** (double lower, double upper) override
- bool **is\_on\_target** () override

### Public Attributes

- **PID pid**

## 5.60.1 Member Function Documentation

### 5.60.1.1 get()

```
double PIDFF::get ( ) [override], [virtual]
```

#### Returns

the last saved result from the feedback controller

Implements [Feedback](#).

### 5.60.1.2 init()

```
void PIDFF::init (
    double start_pt,
    double set_pt,
    double start_vel,
    double end_vel ) [override], [virtual]
```

Initialize the feedback controller for a movement

#### Parameters

<i>start_pt</i>	the current sensor value
<i>set_pt</i>	where the sensor value should be
<i>start_vel</i>	the current rate of change of the sensor value
<i>end_vel</i>	the desired ending rate of change of the sensor value

Initialize the feedback controller for a movement

#### Parameters

<i>start_pt</i>	the current sensor value
<i>set_pt</i>	where the sensor value should be

Implements [Feedback](#).

### 5.60.1.3 is\_on\_target()

```
bool PIDFF::is_on_target ( ) [override], [virtual]
```

#### Returns

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

#### 5.60.1.4 set\_limits()

```
void PIDFF::set_limits (
    double lower,
    double upper ) [override], [virtual]
```

Clamp the upper and lower limits of the output. If both are 0, no limits should be applied.

##### Parameters

<i>lower</i>	Upper limit
<i>upper</i>	Lower limit

Implements [Feedback](#).

#### 5.60.1.5 set\_target()

```
void PIDFF::set_target (
    double set_pt )
```

Set the target of the [PID](#) loop

##### Parameters

<i>set<sub>→</sub> _pt</i>	Setpoint / target value
--------------------------------	-------------------------

#### 5.60.1.6 update() [1/2]

```
double PIDFF::update (
    double val ) [override], [virtual]
```

Iterate the feedback loop once with an updated sensor value. Only kS for feedforward will be applied.

##### Parameters

<i>val</i>	value from the sensor
------------	-----------------------

##### Returns

feedback loop result

Implements [Feedback](#).

#### 5.60.1.7 update() [2/2]

```
double PIDFF::update (
    double val,
```

```
    double vel_setpt,
    double a_setpt = 0 )
```

Iterate the feedback loop once with an updated sensor value

#### Parameters

<i>val</i>	value from the sensor
<i>vel_setpt</i>	Velocity for feedforward
<i>a_setpt</i>	Acceleration for feedforward

#### Returns

feedback loop result

The documentation for this class was generated from the following files:

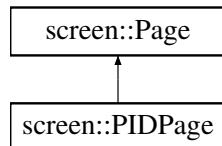
- include/utils/controls/pidff.h
- src/utils/controls/pidff.cpp

## 5.61 screen::PIDPage Class Reference

[PIDPage](#) provides a way to tune a pid controller on the screen.

```
#include <screen.h>
```

Inheritance diagram for screen::PIDPage:



#### Public Member Functions

- [PIDPage](#) (`PID &pid, std::string name, std::function< void(void)> onchange=[ ](){} )`  
Create a [PIDPage](#).
- [PIDPage](#) (`PIDFF &pidff, std::string name, std::function< void(void)> onchange=[ ](){} )`
- `void update (bool was_pressed, int x, int y) override`
- `void draw (vex::brain::lcd &, bool first_draw, unsigned int frame_number) override`

### 5.61.1 Detailed Description

[PIDPage](#) provides a way to tune a pid controller on the screen.

### 5.61.2 Constructor & Destructor Documentation

#### 5.61.2.1 PIDPage()

```
screen::PIDPage::PIDPage (
    PID & pid,
    std::string name,
    std::function< void(void)> onchange = [ ](){} )
```

Create a [PIDPage](#).

## Parameters

<i>pid</i>	the pid controller we're changing
<i>name</i>	a name to recognize this pid controller if we've got multiple pid screens
<i>onchange</i>	a function that is called when a tuning parameter is changed. If you need to update stuff on that change register a handler here

**5.61.3 Member Function Documentation****5.61.3.1 draw()**

```
void screen::PIDPage::draw (
    vex::brain::lcd & scr,
    bool first_draw,
    unsigned int frame_number ) [override], [virtual]
```

## See also

[Page::draw](#)

Reimplemented from [screen::Page](#).

**5.61.3.2 update()**

```
void screen::PIDPage::update (
    bool was_pressed,
    int x,
    int y ) [override], [virtual]
```

## See also

[Page::update](#)

Reimplemented from [screen::Page](#).

The documentation for this class was generated from the following files:

- include/subsystems/screen.h
- src/subsystems/screen.cpp

**5.62 point\_t Struct Reference**

```
#include <geometry.h>
```

## Public Member Functions

- double `dist (const point_t other) const`
- `point_t operator+ (const point_t &other) const`
- `point_t operator- (const point_t &other) const`
- `point_t operator* (double s) const`
- `point_t operator/ (double s) const`
- `point_t operator- () const`
- `point_t operator+ () const`
- bool `operator== (const point_t &rhs)`

## Public Attributes

- double `x`  
*the x position in space*
- double `y`  
*the y position in space*

### 5.62.1 Detailed Description

Data structure representing an X,Y coordinate

### 5.62.2 Member Function Documentation

#### 5.62.2.1 dist()

```
double point_t::dist (
    const point_t other ) const [inline]
```

dist calculates the euclidian distance between this point and another point using the pythagorean theorem

##### Parameters

<code>other</code>	the point to measure the distance from
--------------------	--

##### Returns

the euclidian distance between this and other

#### 5.62.2.2 operator+()

```
point_t point_t::operator+ (
    const point_t & other ) const [inline]
```

Vector2D addition operation on points

**Parameters**

<i>other</i>	the point to add on to this
--------------	-----------------------------

**Returns**

this + other (this.x + other.x, this.y + other.y)

**5.62.2.3 operator-()**

```
point_t point_t::operator- (
    const point_t & other ) const [inline]
```

Vector2D subtraction operation on points

**Parameters**

<i>other</i>	the <a href="#">point_t</a> to subtract from this
--------------	---

**Returns**

this - other (this.x - other.x, this.y - other.y)

The documentation for this struct was generated from the following file:

- [include/utils/geometry.h](#)

**5.63 pose\_t Struct Reference**

```
#include <geometry.h>
```

**Public Member Functions**

- [point\\_t get\\_point \(\)](#)

**Public Attributes**

- double **x**  
*x position in the world*
- double **y**  
*y position in the world*
- double **rot**  
*rotation in the world*

### 5.63.1 Detailed Description

Describes a single position and rotation

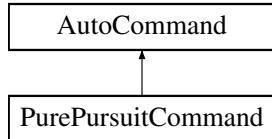
The documentation for this struct was generated from the following file:

- include/utils/geometry.h

## 5.64 PurePursuitCommand Class Reference

```
#include <drive_commands.h>
```

Inheritance diagram for PurePursuitCommand:



### Public Member Functions

- [PurePursuitCommand \(TankDrive &drive\\_sys, Feedback &feedback, PurePursuit::Path path, directionType dir, double max\\_speed=1, double end\\_speed=0\)](#)
- [bool run \(\) override](#)
- [void on\\_timeout \(\) override](#)

### Public Member Functions inherited from [AutoCommand](#)

- [AutoCommand \\* withTimeout \(double t\\_seconds\)](#)
- [AutoCommand \\* withCancelCondition \(Condition \\*true\\_to\\_end\)](#)

### Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- [double timeout\\_seconds = default\\_timeout](#)
- [Condition \\* true\\_to\\_end = nullptr](#)

### Static Public Attributes inherited from [AutoCommand](#)

- [static constexpr double default\\_timeout = 10.0](#)

### 5.64.1 Detailed Description

Autocommand wrapper class for pure pursuit function in the [TankDrive](#) class

## 5.64.2 Constructor & Destructor Documentation

### 5.64.2.1 PurePursuitCommand()

```
PurePursuitCommand::PurePursuitCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    PurePursuit::Path path,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

Construct a Pure Pursuit [AutoCommand](#)

#### Parameters

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>feedback</i>	The feedback controller determining speed
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)

## 5.64.3 Member Function Documentation

### 5.64.3.1 on\_timeout()

```
void PurePursuitCommand::on_timeout ( ) [override], [virtual]
```

Reset the drive system when it times out

Reimplemented from [AutoCommand](#).

### 5.64.3.2 run()

```
bool PurePursuitCommand::run ( ) [override], [virtual]
```

Direct call to [TankDrive::pure\\_pursuit](#)

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command\_structure/drive\_commands.h
- src/utils/command\_structure/drive\_commands.cpp

## 5.65 Rect Struct Reference

### Public Member Functions

- [`point\_t dimensions \(\) const`](#)
- [`point\_t center \(\) const`](#)
- [`double width \(\) const`](#)
- [`double height \(\) const`](#)
- [`bool contains \(point\_t p\) const`](#)

### Static Public Member Functions

- static `Rect from_min_and_size (point_t min, point_t size)`

### Public Attributes

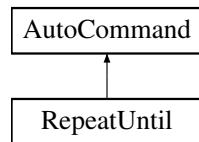
- `point_t min`
- `point_t max`

The documentation for this struct was generated from the following file:

- include/utils/geometry.h

## 5.66 RepeatUntil Class Reference

Inheritance diagram for RepeatUntil:



### Public Member Functions

- `RepeatUntil (InOrder cmd, size_t repeats)`  
*RepeatUntil that runs a fixed number of times.*
- `RepeatUntil (InOrder cmd, Condition *true_to_end)`  
*RepeatUntil the condition.*
- `bool run () override`
- `void on_timeout () override`

### Public Member Functions inherited from `AutoCommand`

- `AutoCommand * withTimeout (double t_seconds)`
- `AutoCommand * withCancelCondition (Condition *true_to_end)`

### Additional Inherited Members

### Public Attributes inherited from `AutoCommand`

- `double timeout_seconds = default_timeout`
- `Condition * true_to_end = nullptr`

### Static Public Attributes inherited from `AutoCommand`

- static constexpr double `default_timeout = 10.0`

## 5.66.1 Constructor & Destructor Documentation

### 5.66.1.1 RepeatUntil() [1/2]

```
RepeatUntil::RepeatUntil (
    InOrder cmds,
    size_t repeats )
```

[RepeatUntil](#) that runs a fixed number of times.

**Parameters**

<i>cmds</i>	the cmds to repeat
<i>repeats</i>	the number of repeats to do

**5.66.1.2 RepeatUntil() [2/2]**

```
RepeatUntil::RepeatUntil (
    InOrder cmds,
    Condition * true_to_end )
```

[RepeatUntil](#) the condition.

**Parameters**

<i>cmds</i>	the cmds to run
<i>true_to_end</i>	we will repeat until <i>true_or_end</i> .test() returns true

**5.66.2 Member Function Documentation****5.66.2.1 on\_timeout()**

```
void RepeatUntil::on_timeout ( ) [override], [virtual]
```

What to do if we timeout instead of finishing. timeout is specified by the timeout seconds in the constructor

Reimplemented from [AutoCommand](#).

**5.66.2.2 run()**

```
bool RepeatUntil::run ( ) [override], [virtual]
```

Executes the command Overridden by child classes

**Returns**

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command\_structure/auto\_command.h
- src/utils/command\_structure/auto\_command.cpp

## 5.67 robot\_specs\_t Struct Reference

```
#include <robot_specs.h>
```

### Public Attributes

- double **robot\_radius**  
*if you were to draw a circle with this radius, the robot would be entirely contained within it*
- double **odom\_wheel\_diam**  
*the diameter of the wheels used for*
- double **odom\_gear\_ratio**  
*the ratio of the odometry wheel to the encoder reading odometry data*
- double **dist\_between\_wheels**  
*the distance between centers of the central drive wheels*
- double **drive\_correction\_cutoff**  
*the distance at which to stop trying to turn towards the target. If we are less than this value, we can continue driving forward to minimize our distance but will not try to spin around to point directly at the target*
- **Feedback \* drive\_feedback**  
*the default feedback for autonomous driving*
- **Feedback \* turn\_feedback**  
*the defualt feedback for autonomous turning*
- **PID::pid\_config\_t correction\_pid**  
*the pid controller to keep the robot driving in as straight a line as possible*

### 5.67.1 Detailed Description

Main robot characterization struct. This will be passed to all the major subsystems that require info about the robot. All distance measurements are in inches.

The documentation for this struct was generated from the following file:

- include/robot\_specs.h

## 5.68 screen::ScreenData Struct Reference

The [ScreenData](#) class holds the data that will be passed to the screen thread you probably shouldnt have to use it.

### Public Member Functions

- **ScreenData** (const std::vector< [Page](#) \* > &m\_pages, int m\_page, vex::brain::lcd &m\_screen)

### Public Attributes

- std::vector< [Page](#) \* > **pages**
- int **page** = 0
- vex::brain::lcd **screen**

### 5.68.1 Detailed Description

The [ScreenData](#) class holds the data that will be passed to the screen thread you probably shouldnt have to use it.

The documentation for this struct was generated from the following file:

- `src/subsystems/screen.cpp`

## 5.69 Serializer Class Reference

Serializes Arbitrary data to a file on the SD Card.

```
#include <serializer.h>
```

### Public Member Functions

- `~Serializer ()`  
*Save and close upon destruction (bc of vex, this doesnt always get called when the program ends. To be sure, call `save_to_disk`)*
- `Serializer (const std::string &filename, bool flush_always=true)`  
*create a `Serializer`*
- `void save_to_disk () const`  
*saves current `Serializer` state to disk*
- `void set_int (const std::string &name, int i)`  
*Setters - not saved until `save_to_disk` is called.*
- `void set_bool (const std::string &name, bool b)`  
*sets a bool by the name of name to b. If `flush_always == true`, this will save to the sd card*
- `void set_double (const std::string &name, double d)`  
*sets a double by the name of name to d. If `flush_always == true`, this will save to the sd card*
- `void set_string (const std::string &name, std::string str)`  
*sets a string by the name of name to s. If `flush_always == true`, this will save to the sd card*
- `int int_or (const std::string &name, int otherwise)`  
*gets a value stored in the serializer. If not found, sets the value to otherwise*
- `bool bool_or (const std::string &name, bool otherwise)`  
*gets a value stored in the serializer. If not, sets the value to otherwise*
- `double double_or (const std::string &name, double otherwise)`  
*gets a value stored in the serializer. If not, sets the value to otherwise*
- `std::string string_or (const std::string &name, std::string otherwise)`  
*gets a value stored in the serializer. If not, sets the value to otherwise*

### 5.69.1 Detailed Description

Serializes Arbitrary data to a file on the SD Card.

### 5.69.2 Constructor & Destructor Documentation

#### 5.69.2.1 Serializer()

```
Serializer::Serializer (
    const std::string & filename,
    bool flush_always = true ) [inline], [explicit]
```

create a `Serializer`

**Parameters**

<i>filename</i>	the file to read from. If filename does not exist we will create that file
<i>flush_always</i>	If true, after every write flush to a file. If false, you are responsible for calling <code>save_to_disk</code>

**5.69.3 Member Function Documentation****5.69.3.1 bool\_or()**

```
bool Serializer::bool_or (
    const std::string & name,
    bool otherwise )
```

gets a value stored in the serializer. If not, sets the value to otherwise

**Parameters**

<i>name</i>	name of value
<i>otherwise</i>	value if the name is not specified

**Returns**

the value if found or otherwise

**5.69.3.2 double\_or()**

```
double Serializer::double_or (
    const std::string & name,
    double otherwise )
```

gets a value stored in the serializer. If not, sets the value to otherwise

**Parameters**

<i>name</i>	name of value
<i>otherwise</i>	value if the name is not specified

**Returns**

the value if found or otherwise

**5.69.3.3 int\_or()**

```
int Serializer::int_or (
    const std::string & name,
    int otherwise )
```

gets a value stored in the serializer. If not found, sets the value to otherwise

Getters Return value if it exists in the serializer

**Parameters**

<i>name</i>	name of value
<i>otherwise</i>	value if the name is not specified

**Returns**

the value if found or otherwise

**5.69.3.4 save\_to\_disk()**

```
void Serializer::save_to_disk ( ) const
saves current Serializer state to disk
forms data bytes then saves to filename this was opened with
```

**5.69.3.5 set\_bool()**

```
void Serializer::set_bool (
    const std::string & name,
    bool b )
```

sets a bool by the name of name to b. If flush\_always == true, this will save to the sd card

**Parameters**

<i>name</i>	name of bool
<i>b</i>	value of bool

**5.69.3.6 set\_double()**

```
void Serializer::set_double (
    const std::string & name,
    double d )
```

sets a double by the name of name to d. If flush\_always == true, this will save to the sd card

**Parameters**

<i>name</i>	name of double
<i>d</i>	value of double

**5.69.3.7 set\_int()**

```
void Serializer::set_int (
```

```
const std::string & name,
int i )
```

Setters - not saved until save\_to\_disk is called.

sets an integer by the name of name to i. If flush\_always == true, this will save to the sd card

#### Parameters

<i>name</i>	name of integer
<i>i</i>	value of integer

### 5.69.3.8 set\_string()

```
void Serializer::set_string (
    const std::string & name,
    std::string str )
```

sets a string by the name of name to s. If flush\_always == true, this will save to the sd card

#### Parameters

<i>name</i>	name of string
<i>i</i>	value of string

### 5.69.3.9 string\_or()

```
std::string Serializer::string_or (
    const std::string & name,
    std::string otherwise )
```

gets a value stored in the serializer. If not, sets the value to otherwise

#### Parameters

<i>name</i>	name of value
<i>otherwise</i>	value if the name is not specified

#### Returns

the value if found or otherwise

The documentation for this class was generated from the following files:

- include/utils/serializer.h
- src/utils/serializer.cpp

## 5.70 screen::SliderWidget Class Reference

Widget that updates a double value. Updates by reference so watch out for race conditions cuz the screen stuff lives on another thread.

```
#include <screen.h>
```

### Public Member Functions

- **SliderWidget** (double &val, double low, double high, **Rect** rect, std::string name)  
*Creates a slider widget.*
- bool **update** (bool was\_pressed, int x, int y)  
*responds to user input*
- void **draw** (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number)  
*Page::draws the slide to the screen*

### 5.70.1 Detailed Description

Widget that updates a double value. Updates by reference so watch out for race conditions cuz the screen stuff lives on another thread.

### 5.70.2 Constructor & Destructor Documentation

#### 5.70.2.1 SliderWidget()

```
screen::SliderWidget::SliderWidget (
    double & val,
    double low,
    double high,
    Rect rect,
    std::string name ) [inline]
```

Creates a slider widget.

#### Parameters

<i>val</i>	reference to the value to modify
<i>low</i>	minimum value to go to
<i>high</i>	maximum value to go to
<i>rect</i>	rect to draw it
<i>name</i>	name of the value

### 5.70.3 Member Function Documentation

#### 5.70.3.1 update()

```
bool screen::SliderWidget::update (
```

```
    bool was_pressed,
    int x,
    int y )
```

responds to user input

#### Parameters

<code>was_pressed</code>	if the screen is pressed
<code>x</code>	x position if the screen was pressed
<code>y</code>	y position if the screen was pressed

#### Returns

true if the value updated

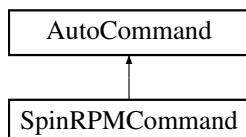
The documentation for this class was generated from the following files:

- include/subsystems/screen.h
- src/subsystems/screen.cpp

## 5.71 SpinRPMCommand Class Reference

```
#include <flywheel_commands.h>
```

Inheritance diagram for SpinRPMCommand:



#### Public Member Functions

- [SpinRPMCommand \(Flywheel &flywheel, int rpm\)](#)
- [bool run \(\) override](#)

#### Public Member Functions inherited from [AutoCommand](#)

- [virtual void on\\_timeout \(\)](#)
- [AutoCommand \\* withTimeout \(double t\\_seconds\)](#)
- [AutoCommand \\* withCancelCondition \(Condition \\*true\\_to\\_end\)](#)

#### Additional Inherited Members

#### Public Attributes inherited from [AutoCommand](#)

- [double timeout\\_seconds = default\\_timeout](#)
- [Condition \\* true\\_to\\_end = nullptr](#)

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default\_timeout** = 10.0

### 5.71.1 Detailed Description

File: [flywheel\\_commands.h](#) Desc: [insert meaningful desc] [AutoCommand](#) wrapper class for the spin\_rpm function in the [Flywheel](#) class

### 5.71.2 Constructor & Destructor Documentation

#### 5.71.2.1 SpinRPMCommand()

```
SpinRPMCommand::SpinRPMCommand (
    Flywheel & flywheel,
    int rpm )
```

Construct a SpinRPM Command

##### Parameters

<i>flywheel</i>	the flywheel sys to command
<i>rpm</i>	the rpm that we should spin at

File: [flywheel\\_commands.cpp](#) Desc: [insert meaningful desc]

### 5.71.3 Member Function Documentation

#### 5.71.3.1 run()

```
bool SpinRPMCommand::run ( ) [override], [virtual]
```

Run spin\_manual Overrides run from [AutoCommand](#)

##### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command\_structure/flywheel\_commands.h
- src/utils/command\_structure/flywheel\_commands.cpp

## 5.72 PurePursuit::spline Struct Reference

```
#include <pure_pursuit.h>
```

### Public Member Functions

- double **getY** (double x)

### Public Attributes

- double **a**
- double **b**
- double **c**
- double **d**
- double **x\_start**
- double **x\_end**

### 5.72.1 Detailed Description

Represents a piece of a cubic spline with  $s(x) = a(x-x_i)^3 + b(x-x_i)^2 + c(x-x_i) + d$ . The **x\_start** and **x\_end** shows where the equation is valid.

The documentation for this struct was generated from the following file:

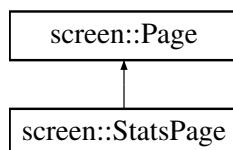
- include/utils/pure\_pursuit.h

## 5.73 screen::StatsPage Class Reference

Draws motor stats and battery stats to the screen.

```
#include <screen.h>
```

Inheritance diagram for screen::StatsPage:



### Public Member Functions

- **StatsPage** (std::map< std::string, vex::motor & > motors)  
*Creates a stats page.*
- void **update** (bool was\_pressed, int x, int y) override
- void **draw** (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number) override

### 5.73.1 Detailed Description

Draws motor stats and battery stats to the screen.

### 5.73.2 Constructor & Destructor Documentation

#### 5.73.2.1 StatsPage()

```
screen::StatsPage::StatsPage (
    std::map< std::string, vex::motor & > motors )
```

Creates a stats page.

##### Parameters

<i>motors</i>	a map of string to motor that we want to draw on this page
---------------	--

### 5.73.3 Member Function Documentation

#### 5.73.3.1 draw()

```
void screen::StatsPage::draw (
    vex::brain::lcd & scr,
    bool first_draw,
    unsigned int frame_number ) [override], [virtual]
```

##### See also

[Page::draw](#)

Reimplemented from [screen::Page](#).

#### 5.73.3.2 update()

```
void screen::StatsPage::update (
    bool was_pressed,
    int x,
    int y ) [override], [virtual]
```

##### See also

[Page::update](#)

Reimplemented from [screen::Page](#).

The documentation for this class was generated from the following files:

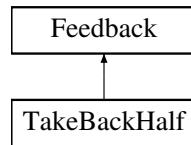
- include/subsystems/screen.h
- src/subsystems/screen.cpp

## 5.74 TakeBackHalf Class Reference

A velocity controller.

```
#include <take_back_half.h>
```

Inheritance diagram for TakeBackHalf:



### Public Member Functions

- **TakeBackHalf** (double `TBH_gain`, double `first_cross_split`, double `on_target_threshold`)
- void `init` (double `start_pt`, double `set_pt`, double, double)
- double `update` (double `val`) override
- double `get` () override
- void `set_limits` (double `lower`, double `upper`) override
- bool `is_on_target` () override

### Public Attributes

- double `TBH_gain`  
*tuned parameter*
- double `first_cross_split`

### 5.74.1 Detailed Description

A velocity controller.

#### Warning

If you try to use this as a position controller, it will fail.

### 5.74.2 Member Function Documentation

#### 5.74.2.1 `get()`

```
double TakeBackHalf::get ( ) [override], [virtual]
```

#### Returns

the last saved result from the feedback controller

Implements [Feedback](#).

#### 5.74.2.2 `init()`

```
void TakeBackHalf::init (
    double start_pt,
    double set_pt,
    double ,
    double ) [virtual]
```

Initialize the feedback controller for a movement

**Parameters**

<i>start_pt</i>	the current sensor value
<i>set_pt</i>	where the sensor value should be
<i>start_vel</i>	Movement starting velocity (IGNORED)
<i>end_vel</i>	Movement ending velocity (IGNORED)

Implements [Feedback](#).

**5.74.2.3 is\_on\_target()**

```
bool TakeBackHalf::is_on_target ( ) [override], [virtual]
```

**Returns**

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

**5.74.2.4 set\_limits()**

```
void TakeBackHalf::set_limits (
    double lower,
    double upper ) [override], [virtual]
```

Clamp the upper and lower limits of the output. If both are 0, no limits should be applied.

**Parameters**

<i>lower</i>	Upper limit
<i>upper</i>	Lower limit

Implements [Feedback](#).

**5.74.2.5 update()**

```
double TakeBackHalf::update (
    double val ) [override], [virtual]
```

Iterate the feedback loop once with an updated sensor value

**Parameters**

<i>val</i>	value from the sensor
------------	-----------------------

**Returns**

feedback loop result

Implements [Feedback](#).

The documentation for this class was generated from the following files:

- include/utils/controls/take\_back\_half.h
- src/utils/controls/take\_back\_half.cpp

## 5.75 TankDrive Class Reference

```
#include <tank_drive.h>
```

### Public Member Functions

- [`TankDrive`](#) (`motor_group &left_motors, motor_group &right_motors, robot_specs_t &config, OdometryBase *odom=NULL`)
- [`AutoCommand \* DriveToPointCmd`](#) (`point_t pt, vex::directionType dir=vex::forward, double max_speed=1.0, double end_speed=0.0`)
- [`AutoCommand \* DriveToPointCmd`](#) (`Feedback &fb, point_t pt, vex::directionType dir=vex::forward, double max_speed=1.0, double end_speed=0.0`)
- [`AutoCommand \* DriveForwardCmd`](#) (`double dist, vex::directionType dir=vex::forward, double max_speed=1.0, double end_speed=0.0`)
- [`AutoCommand \* DriveForwardCmd`](#) (`Feedback &fb, double dist, vex::directionType dir=vex::forward, double max_speed=1.0, double end_speed=0.0`)
- [`AutoCommand \* TurnToHeadingCmd`](#) (`double heading, double max_speed=1.0, double end_speed=0.0`)
- [`AutoCommand \* TurnToHeadingCmd`](#) (`Feedback &fb, double heading, double max_speed=1.0, double end_speed=0.0`)
- [`AutoCommand \* TurnDegreesCmd`](#) (`double degrees, double max_speed=1.0, double start_speed=0.0`)
- [`AutoCommand \* TurnDegreesCmd`](#) (`Feedback &fb, double degrees, double max_speed=1.0, double end_speed=0.0`)
- [`AutoCommand \* PurePursuitCmd`](#) (`PurePursuit::Path path, directionType dir, double max_speed=1, double end_speed=0`)
- [`AutoCommand \* PurePursuitCmd`](#) (`Feedback &feedback, PurePursuit::Path path, directionType dir, double max_speed=1, double end_speed=0`)
- `void stop ()`
- `void drive_tank (double left, double right, int power=1)`
- `void drive_arcade (double forward_back, double left_right, int power=1)`
- `bool drive_forward (double inches, directionType dir, Feedback &feedback, double max_speed=1, double end_speed=0)`
- `bool drive_forward (double inches, directionType dir, double max_speed=1, double end_speed=0)`
- `bool turn_degrees (double degrees, Feedback &feedback, double max_speed=1, double end_speed=0)`
- `bool turn_degrees (double degrees, double max_speed=1, double end_speed=0)`
- `bool drive_to_point (double x, double y, vex::directionType dir, Feedback &feedback, double max_speed=1, double end_speed=0)`
- `bool drive_to_point (double x, double y, vex::directionType dir, double max_speed=1, double end_speed=0)`
- `bool turn_to_heading (double heading_deg, Feedback &feedback, double max_speed=1, double end_speed=0)`
- `bool turn_to_heading (double heading_deg, double max_speed=1, double end_speed=0)`
- `void reset_auto ()`
- `bool pure_pursuit (PurePursuit::Path path, directionType dir, Feedback &feedback, double max_speed=1, double end_speed=0)`
- `bool pure_pursuit (PurePursuit::Path path, directionType dir, double max_speed=1, double end_speed=0)`

### Static Public Member Functions

- static double [modify\\_inputs](#) (double input, int power=2)

### 5.75.1 Detailed Description

[TankDrive](#) is a class to run a tank drive system. A tank drive system, sometimes called differential drive, has a motor (or group of synchronized motors) on the left and right side

### 5.75.2 Constructor & Destructor Documentation

#### 5.75.2.1 [TankDrive\(\)](#)

```
TankDrive:::TankDrive (
    motor_group & left_motors,
    motor_group & right_motors,
    robot_specs_t & config,
    OdometryBase * odom = NULL )
```

Create the [TankDrive](#) object

#### Parameters

<i>left_motors</i>	left side drive motors
<i>right_motors</i>	right side drive motors
<i>config</i>	the configuration specification defining physical dimensions about the robot. See <a href="#">robot_specs_t</a> for more info
<i>odom</i>	an odometry system to track position and rotation. this is necessary to execute autonomous paths

### 5.75.3 Member Function Documentation

#### 5.75.3.1 [drive\\_arCADE\(\)](#)

```
void TankDrive:::drive_arCADE (
    double forward_back,
    double left_right,
    int power = 1 )
```

Drive the robot using arcade style controls. *forward\_back* controls the linear motion, *left\_right* controls the turning.

*forward\_back* and *left\_right* are in "percent": -1.0 -> 1.0

#### Parameters

<i>forward_back</i>	the percent to move forward or backward
<i>left_right</i>	the percent to turn left or right
<i>power</i>	modifies the input velocities left^power, right^power

Drive the robot using arcade style controls. forward\_back controls the linear motion, left\_right controls the turning.

left\_motors and right\_motors are in "percent": -1.0 -> 1.0

### 5.75.3.2 drive\_forward() [1/2]

```
bool TankDrive::drive_forward (
    double inches,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

Autonomously drive the robot forward a certain distance

#### Parameters

<i>inches</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>dir</i>	the direction we want to travel forward and backward
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Autonomously drive the robot forward a certain distance

#### Parameters

<i>inches</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>dir</i>	the direction we want to travel forward and backward
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

#### Returns

true if we have finished driving to our point

### 5.75.3.3 drive\_forward() [2/2]

```
bool TankDrive::drive_forward (
    double inches,
    directionType dir,
    Feedback & feedback,
    double max_speed = 1,
    double end_speed = 0 )
```

Use odometry to drive forward a certain distance using a custom feedback controller

Returns whether or not the robot has reached it's destination.

#### Parameters

<i>inches</i>	the distance to drive forward
---------------	-------------------------------

**Parameters**

<i>dir</i>	the direction we want to travel forward and backward
<i>feedback</i>	the custom feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

**Returns**

true when we have reached our target distance

Use odometry to drive forward a certain distance using a custom feedback controller

Returns whether or not the robot has reached it's destination.

**Parameters**

<i>inches</i>	the distance to drive forward
<i>dir</i>	the direction we want to travel forward and backward
<i>feedback</i>	the custom feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

**5.75.3.4 drive\_tank()**

```
void TankDrive::drive_tank (
    double left,
    double right,
    int power = 1 )
```

Drive the robot using differential style controls. *left\_motors* controls the left motors, *right\_motors* controls the right motors.

*left\_motors* and *right\_motors* are in "percent": -1.0 -> 1.0

**Parameters**

<i>left</i>	the percent to run the left motors
<i>right</i>	the percent to run the right motors
<i>power</i>	modifies the input velocities <i>left</i> <sup><i>power</i></sup> , <i>right</i> <sup><i>power</i></sup>
<i>isdriver</i>	default false. if true uses motor percentage. if false uses plain percentage of maximum voltage

Drive the robot using differential style controls. *left\_motors* controls the left motors, *right\_motors* controls the right motors.

*left\_motors* and *right\_motors* are in "percent": -1.0 -> 1.0

### 5.75.3.5 drive\_to\_point() [1/2]

```
bool TankDrive::drive_to_point (
    double x,
    double y,
    vex::directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot. Here we use the default feedback controller from the drive\_sys

Returns whether or not the robot has reached it's destination.

#### Parameters

<i>x</i>	the x position of the target
<i>y</i>	the y position of the target
<i>dir</i>	the direction we want to travel forward and backward
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot. Here we use the default feedback controller from the drive\_sys

Returns whether or not the robot has reached it's destination.

#### Parameters

<i>x</i>	the x position of the target
<i>y</i>	the y position of the target
<i>dir</i>	the direction we want to travel forward and backward
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

#### Returns

true if we have reached our target point

### 5.75.3.6 drive\_to\_point() [2/2]

```
bool TankDrive::drive_to_point (
    double x,
    double y,
    vex::directionType dir,
    Feedback & feedback,
    double max_speed = 1,
    double end_speed = 0 )
```

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot.

Returns whether or not the robot has reached it's destination.

**Parameters**

<i>x</i>	the x position of the target
<i>y</i>	the y position of the target
<i>dir</i>	the direction we want to travel forward and backward
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot.

Returns whether or not the robot has reached it's destination.

**Parameters**

<i>x</i>	the x position of the target
<i>y</i>	the y position of the target
<i>dir</i>	the direction we want to travel forward and backward
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

**Returns**

true if we have reached our target point

**5.75.3.7 modify\_inputs()**

```
double TankDrive::modify_inputs (
    double input,
    int power = 2 ) [static]
```

Create a curve for the inputs, so that drivers have more control at lower speeds. Curves are exponential, with the default being squaring the inputs.

**Parameters**

<i>input</i>	the input before modification
<i>power</i>	the power to raise input to

**Returns**

$\text{input}^{\wedge} \text{power}$  (accounts for negative inputs and odd numbered powers)

Modify the inputs from the controller by squaring / cubing, etc Allows for better control of the robot at slower speeds

**Parameters**

<i>input</i>	the input signal -1 -> 1
<i>power</i>	the power to raise the signal to

**Returns**

input^power accounting for any sign issues that would arise with this naive solution

**5.75.3.8 pure\_pursuit() [1/2]**

```
bool TankDrive::pure_pursuit (
    PurePursuit::Path path,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0 )
```

Drive the robot autonomously using a pure-pursuit algorithm - Input path with a set of waypoints - the robot will attempt to follow the points while cutting corners (radius) to save time (compared to stop / turn / start)

Use the default drive feedback

**Parameters**

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

**Returns**

True when the path is complete

Drive the robot autonomously using a pure-pursuit algorithm - Input path with a set of waypoints - the robot will attempt to follow the points while cutting corners (radius) to save time (compared to stop / turn / start)

Use the default drive feedback

**Parameters**

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)

**Returns**

True when the path is complete

**5.75.3.9 pure\_pursuit() [2/2]**

```
bool TankDrive::pure_pursuit (
    PurePursuit::Path path,
    directionType dir,
    Feedback & feedback,
```

```
double max_speed = 1,
double end_speed = 0 )
```

Drive the robot autonomously using a pure-pursuit algorithm - Input path with a set of waypoints - the robot will attempt to follow the points while cutting corners (radius) to save time (compared to stop / turn / start)

#### Parameters

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>feedback</i>	The feedback controller determining speed
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

#### Returns

True when the path is complete

Drive the robot autonomously using a pure-pursuit algorithm - Input path with a set of waypoints - the robot will attempt to follow the points while cutting corners (radius) to save time (compared to stop / turn / start)

#### Parameters

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>feedback</i>	The feedback controller determining speed
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)

#### Returns

True when the path is complete

### 5.75.3.10 reset\_auto()

```
void TankDrive::reset_auto ( )
```

Reset the initialization for autonomous drive functions

### 5.75.3.11 stop()

```
void TankDrive::stop ( )
```

Stops rotation of all the motors using their "brake mode"

**5.75.3.12 turn\_degrees() [1/2]**

```
bool TankDrive::turn_degrees (
    double degrees,
    double max_speed = 1,
    double end_speed = 0 )
```

Autonomously turn the robot X degrees to counterclockwise (negative for clockwise), with a maximum motor speed of percent\_speed (-1.0 -> 1.0)

Uses the default turning feedback of the drive system.

## Parameters

<i>degrees</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Autonomously turn the robot X degrees to counterclockwise (negative for clockwise), with a maximum motor speed of percent\_speed (-1.0 -> 1.0)

Uses the default turning feedback of the drive system.

## Parameters

<i>degrees</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

## Returns

true if we turned to target number of degrees

**5.75.3.13 turn\_degrees() [2/2]**

```
bool TankDrive::turn_degrees (
    double degrees,
    Feedback & feedback,
    double max_speed = 1,
    double end_speed = 0 )
```

Autonomously turn the robot X degrees counterclockwise (negative for clockwise), with a maximum motor speed of percent\_speed (-1.0 -> 1.0)

Uses PID + Feedforward for its control.

## Parameters

<i>degrees</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power

Autonomously turn the robot X degrees to counterclockwise (negative for clockwise), with a maximum motor speed of percent\_speed (-1.0 -> 1.0)

Uses the specified feedback for its control.

## Parameters

<i>degrees</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

**Returns**

true if we have turned our target number of degrees

**5.75.3.14 turn\_to\_heading() [1/2]**

```
bool TankDrive::turn_to_heading (
    double heading_deg,
    double max_speed = 1,
    double end_speed = 0 )
```

Turn the robot in place to an exact heading relative to the field. 0 is forward. Uses the defualt turn feedback of the drive system

**Parameters**

<i>heading_deg</i>	the heading to which we will turn
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Turn the robot in place to an exact heading relative to the field. 0 is forward. Uses the defualt turn feedback of the drive system

**Parameters**

<i>heading_deg</i>	the heading to which we will turn
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

**Returns**

true if we have reached our target heading

**5.75.3.15 turn\_to\_heading() [2/2]**

```
bool TankDrive::turn_to_heading (
    double heading_deg,
    Feedback & feedback,
    double max_speed = 1,
    double end_speed = 0 )
```

Turn the robot in place to an exact heading relative to the field. 0 is forward.

**Parameters**

<i>heading_deg</i>	the heading to which we will turn
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Turn the robot in place to an exact heading relative to the field. 0 is forward.

#### Parameters

<i>heading_deg</i>	the heading to which we will turn
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

#### Returns

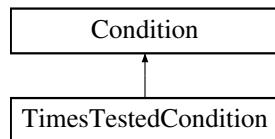
true if we have reached our target heading

The documentation for this class was generated from the following files:

- include/subsystems/tank\_drive.h
- src/subsystems/tank\_drive.cpp

## 5.76 TimesTestedCondition Class Reference

Inheritance diagram for TimesTestedCondition:



#### Public Member Functions

- **TimesTestedCondition** (size\_t N)
- bool **test** () override

#### Public Member Functions inherited from [Condition](#)

- [Condition \\* Or](#) ([Condition](#) \*b)
- [Condition \\* And](#) ([Condition](#) \*b)

### 5.76.1 Member Function Documentation

#### 5.76.1.1 test()

```
bool TimesTestedCondition::test ( ) [inline], [override], [virtual]
```

Implements [Condition](#).

The documentation for this class was generated from the following file:

- include/utils/command\_structure/auto\_command.h

## 5.77 trapezoid\_profile\_segment\_t Struct Reference

```
#include <trapezoid_profile.h>
```

### Public Attributes

- double **pos\_after**  
*1d position after this segment concludes*
- double **vel\_after**  
*1d velocity after this segment concludes*
- double **accel**  
*1d acceleration during the segment*
- double **duration**  
*duration of the segment*

### 5.77.1 Detailed Description

[trapezoid\\_profile\\_segment\\_t](#) is a description of one constant acceleration segment of a trapezoid motion profile

The documentation for this struct was generated from the following file:

- include/utils/controls/trapezoid\_profile.h

## 5.78 TrapezoidProfile Class Reference

```
#include <trapezoid_profile.h>
```

### Public Member Functions

- [TrapezoidProfile](#) (double max\_v, double accel)  
*Construct a new Trapezoid Profile object.*
- [motion\\_t calculate](#) (double time\_s, double pos\_s)  
*Run the trapezoidal profile based on the time and distance that's elapsed.*
- [motion\\_t calculate\\_time\\_based](#) (double time\_s)  
*Run the trapezoidal profile based on the time that's elapsed.*
- void [set\\_endpts](#) (double start, double end)  
*set\_endpts defines a start and end position*
- void [set\\_vel\\_endpts](#) (double start, double end)  
*set start and end velocities*
- void [set\\_accel](#) (double accel)  
*set\_accel sets the acceleration this profile will use (the left and right legs of the trapezoid)*
- void [set\\_max\\_v](#) (double max\_v)  
*sets the maximum velocity for the profile (the height of the top of the trapezoid)*
- double [get\\_movement\\_time](#) ()  
*uses the kinematic equations to and specified accel and max\_v to figure out how long moving along the profile would take*

## 5.78.1 Detailed Description

Trapezoid Profile

This is a motion profile defined by:

- maximum acceleration
- maximum velocity
- start position and velocity
- end position and velocity

Using this information, a parametric function is generated, with a period of acceleration, constant velocity, and deceleration. The velocity graph usually looks like a trapezoid, giving it its name.

If the maximum velocity is set high enough, this will become a S-curve profile, with only acceleration and deceleration.

If the initial velocity is in the wrong direction, the profile will first come to a stop, then continue a normal trapezoid profile.

If the initial velocity is higher than the maximum velocity, the profile will first try to achieve the maximum velocity.

If the end velocity is not achievable, the profile will try to get as close as possible. The end velocity must be in the direction of the end point.

This class is designed for use in properly modelling the motion of the robots to create a feedforward and target for **PID**. Acceleration and Maximum velocity should be measured on the robot and tuned down slightly to account for battery drop.

Here are the equations graphed for ease of understanding: <https://www.desmos.com/calculator/rkm3ivulyk>

### Author

Ryan McGee

### Date

7/12/2022

## 5.78.2 Constructor & Destructor Documentation

### 5.78.2.1 TrapezoidProfile()

```
TrapezoidProfile::TrapezoidProfile (
    double max_v,
    double accel )
```

Construct a new Trapezoid Profile object.

**Parameters**

<i>max_v</i>	Maximum velocity the robot can run at
<i>accel</i>	Maximum acceleration of the robot

**5.78.3 Member Function Documentation****5.78.3.1 calculate()**

```
motion_t TrapezoidProfile::calculate (
    double time_s,
    double pos_s )
```

Run the trapezoidal profile based on the time and distance that's elapsed.

**Parameters**

<i>time_s</i>	Time since start of movement
<i>pos_s</i>	The current position

**Returns**

*motion\_t* Position, velocity and acceleration

**5.78.3.2 calculate\_time\_based()**

```
motion_t TrapezoidProfile::calculate_time_based (
    double time_s )
```

Run the trapezoidal profile based on the time that's elapsed.

**Parameters**

<i>time_s</i>	Time since start of movement
---------------	------------------------------

**Returns**

*motion\_t* Position, velocity and acceleration

**5.78.3.3 get\_movement\_time()**

```
double TrapezoidProfile::get_movement_time ( )
```

uses the kinematic equations to and specified accel and max\_v to figure out how long moving along the profile would take

**Returns**

the time the path will take to travel

**5.78.3.4 set\_accel()**

```
void TrapezoidProfile::set_accel (
    double accel )
```

set\_accel sets the acceleration this profile will use (the left and right legs of the trapezoid)

**Parameters**

<i>accel</i>	the acceleration amount to use
--------------	--------------------------------

**5.78.3.5 set\_endpts()**

```
void TrapezoidProfile::set_endpts (
    double start,
    double end )
```

set\_endpts defines a start and end position

**Parameters**

<i>start</i>	the starting position of the path
<i>end</i>	the ending position of the path

**5.78.3.6 set\_max\_v()**

```
void TrapezoidProfile::set_max_v (
    double max_v )
```

sets the maximum velocity for the profile (the height of the top of the trapezoid)

**Parameters**

<i>max_v</i>	the maximum velocity the robot can travel at
--------------	--

**5.78.3.7 set\_vel\_endpts()**

```
void TrapezoidProfile::set_vel_endpts (
    double start,
    double end )
```

set start and end velocities

**Parameters**

<i>start</i>	the starting velocity of the path
<i>end</i>	the ending velocity of the path

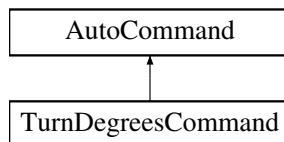
The documentation for this class was generated from the following files:

- include/utils/controls/trapezoid\_profile.h
- src/utils/trapezoid\_profile.cpp

## 5.79 TurnDegreesCommand Class Reference

```
#include <drive_commands.h>
```

Inheritance diagram for TurnDegreesCommand:



### Public Member Functions

- `TurnDegreesCommand (TankDrive &drive_sys, Feedback &feedback, double degrees, double max_speed=1, double end_speed=0)`
- `bool run () override`
- `void on_timeout () override`

### Public Member Functions inherited from [AutoCommand](#)

- `AutoCommand * withTimeout (double t_seconds)`
- `AutoCommand * withCancelCondition (Condition *true_to_end)`

### Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- `double timeout_seconds = default_timeout`
- `Condition * true_to_end = nullptr`

### Static Public Attributes inherited from [AutoCommand](#)

- `static constexpr double default_timeout = 10.0`

## 5.79.1 Detailed Description

[AutoCommand](#) wrapper class for the turn\_degrees function in the [TankDrive](#) class

## 5.79.2 Constructor & Destructor Documentation

### 5.79.2.1 TurnDegreesCommand()

```
TurnDegreesCommand::TurnDegreesCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    double degrees,
    double max_speed = 1,
    double end_speed = 0 )
```

Construct a [TurnDegreesCommand](#) Command

#### Parameters

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the turn
<i>degrees</i>	how many degrees to rotate
<i>max_speed</i>	0 -> 1 percentage of the drive systems speed to drive at

## 5.79.3 Member Function Documentation

### 5.79.3.1 on\_timeout()

```
void TurnDegreesCommand::on_timeout ( ) [override], [virtual]
```

Cleans up drive system if we time out before finishing

reset the drive system if we timeout

Reimplemented from [AutoCommand](#).

### 5.79.3.2 run()

```
bool TurnDegreesCommand::run ( ) [override], [virtual]
```

Run turn\_degrees Overrides run from [AutoCommand](#)

#### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

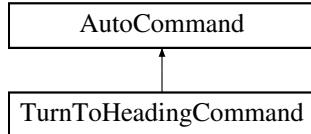
The documentation for this class was generated from the following files:

- include/utils/command\_structure/drive\_commands.h
- src/utils/command\_structure/drive\_commands.cpp

## 5.80 TurnToHeadingCommand Class Reference

```
#include <drive_commands.h>
```

Inheritance diagram for TurnToHeadingCommand:



### Public Member Functions

- `TurnToHeadingCommand (TankDrive &drive_sys, Feedback &feedback, double heading_deg, double speed=1, double end_speed=0)`
- `bool run () override`
- `void on_timeout () override`

### Public Member Functions inherited from [AutoCommand](#)

- `AutoCommand * withTimeout (double t_seconds)`
- `AutoCommand * withCancelCondition (Condition *true_to_end)`

### Additional Inherited Members

#### Public Attributes inherited from [AutoCommand](#)

- `double timeout_seconds = default_timeout`
- `Condition * true_to_end = nullptr`

#### Static Public Attributes inherited from [AutoCommand](#)

- `static constexpr double default_timeout = 10.0`

### 5.80.1 Detailed Description

[AutoCommand](#) wrapper class for the turn\_to\_heading() function in the [TankDrive](#) class

### 5.80.2 Constructor & Destructor Documentation

#### 5.80.2.1 [TurnToHeadingCommand\(\)](#)

```
TurnToHeadingCommand::TurnToHeadingCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    double heading_deg,
    double max_speed = 1,
    double end_speed = 0 )
```

Construct a [TurnToHeadingCommand](#) Command

**Parameters**

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the drive
<i>heading_deg</i>	the heading to turn to in degrees
<i>max_speed</i>	0 -> 1 percentage of the drive systems speed to drive at

**5.80.3 Member Function Documentation****5.80.3.1 on\_timeout()**

```
void TurnToHeadingCommand::on_timeout ( ) [override], [virtual]
```

Cleans up drive system if we time out before finishing

reset the drive system if we don't hit our target

Reimplemented from [AutoCommand](#).

**5.80.3.2 run()**

```
bool TurnToHeadingCommand::run ( ) [override], [virtual]
```

Run turn\_to\_heading Overrides run from [AutoCommand](#)

**Returns**

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command\_structure/drive\_commands.h
- src/utils/command\_structure/drive\_commands.cpp

**5.81 Vector2D Class Reference**

```
#include <vector2d.h>
```

## Public Member Functions

- [Vector2D \(double dir, double mag\)](#)
- [Vector2D \(point\\_t p\)](#)
- [double get\\_dir \(\) const](#)
- [double get\\_mag \(\) const](#)
- [double get\\_x \(\) const](#)
- [double get\\_y \(\) const](#)
- [Vector2D normalize \(\)](#)
- [point\\_t point \(\)](#)
- [Vector2D operator\\* \(const double &x\)](#)
- [Vector2D operator+ \(const Vector2D &other\)](#)
- [Vector2D operator- \(const Vector2D &other\)](#)

### 5.81.1 Detailed Description

`Vector2D` is an x,y pair Used to represent 2D locations on the field. It can also be treated as a direction and magnitude

### 5.81.2 Constructor & Destructor Documentation

#### 5.81.2.1 `Vector2D()` [1/2]

```
Vector2D::Vector2D (
    double dir,
    double mag )
```

Construct a vector object.

##### Parameters

<code>dir</code>	Direction, in radians. 'foward' is 0, clockwise positive when viewed from the top.
<code>mag</code>	Magnitude.

#### 5.81.2.2 `Vector2D()` [2/2]

```
Vector2D::Vector2D (
    point_t p )
```

Construct a vector object from a cartesian point.

##### Parameters

<code>p</code>	<code>point_t.x</code> , <code>point_t.y</code>
----------------	---

### 5.81.3 Member Function Documentation

#### 5.81.3.1 get\_dir()

```
double Vector2D::get_dir ( ) const
```

Get the direction of the vector, in radians. '0' is forward, clockwise positive when viewed from the top.

Use r2d() to convert.

##### Returns

the direction of the vector in radians

Get the direction of the vector, in radians. '0' is forward, clockwise positive when viewed from the top.

Use r2d() to convert.

#### 5.81.3.2 get\_mag()

```
double Vector2D::get_mag ( ) const
```

##### Returns

the magnitude of the vector

Get the magnitude of the vector

#### 5.81.3.3 get\_x()

```
double Vector2D::get_x ( ) const
```

##### Returns

the X component of the vector; positive to the right.

Get the X component of the vector; positive to the right.

#### 5.81.3.4 get\_y()

```
double Vector2D::get_y ( ) const
```

##### Returns

the Y component of the vector, positive forward.

Get the Y component of the vector, positive forward.

### 5.81.3.5 normalize()

```
Vector2D Vector2D::normalize ( )
```

Changes the magnitude of the vector to 1

#### Returns

the normalized vector

Changes the magnetude of the vector to 1

### 5.81.3.6 operator\*()

```
Vector2D Vector2D::operator* (
    const double & x )
```

Scales a `Vector2D` by a scalar with the `*` operator

#### Parameters

<code>x</code>	the value to scale the vector by
----------------	----------------------------------

#### Returns

the this `Vector2D` scaled by x

### 5.81.3.7 operator+()

```
Vector2D Vector2D::operator+ (
    const Vector2D & other )
```

Add the components of two vectors together `Vector2D + Vector2D = (this.x + other.x, this.y + other.y)`

#### Parameters

<code>other</code>	the vector to add to this
--------------------	---------------------------

#### Returns

the sum of the vectors

### 5.81.3.8 operator-()

```
Vector2D Vector2D::operator- (
    const Vector2D & other )
```

Subtract the components of two vectors together `Vector2D - Vector2D = (this.x - other.x, this.y - other.y)`

**Parameters**

<i>other</i>	the vector to subtract from this
--------------	----------------------------------

**Returns**

the difference of the vectors

**5.81.3.9 point()**

`point_t Vector2D::point ( )`

Returns a point from the vector

**Returns**

the point represented by the vector

Convert a direction and magnitude representation to an x, y representation

**Returns**

the x, y representation of the vector

The documentation for this class was generated from the following files:

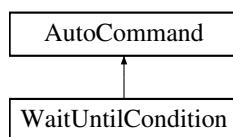
- `include/utils/vector2d.h`
- `src/utils/vector2d.cpp`

## 5.82 WaitUntilCondition Class Reference

Waits until the condition is true.

```
#include <auto_command.h>
```

Inheritance diagram for WaitUntilCondition:



### Public Member Functions

- `WaitUntilCondition (Condition *cond)`
- `bool run () override`

## Public Member Functions inherited from [AutoCommand](#)

- virtual void [on\\_timeout \(\)](#)
- [AutoCommand \\* withTimeout](#) (double t\_seconds)
- [AutoCommand \\* withCancelCondition](#) ([Condition](#) \*true\_to\_end)

## Additional Inherited Members

### Public Attributes inherited from [AutoCommand](#)

- double [timeout\\_seconds](#) = default\_timeout
- [Condition \\* true\\_to\\_end](#) = nullptr

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

## 5.82.1 Detailed Description

Waits until the condition is true.

## 5.82.2 Member Function Documentation

### 5.82.2.1 run()

```
bool WaitUntilCondition::run ( ) [inline], [override], [virtual]
```

Executes the command Overridden by child classes

#### Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

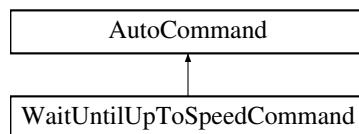
The documentation for this class was generated from the following file:

- include/utils/command\_structure/auto\_command.h

## 5.83 WaitUntilUpToSpeedCommand Class Reference

```
#include <flywheel_commands.h>
```

Inheritance diagram for WaitUntilUpToSpeedCommand:



## Public Member Functions

- [WaitUntilUpToSpeedCommand \(`Flywheel` &`flywheel`, int `threshold\_rpm`\)](#)
- bool `run ()` override

## Public Member Functions inherited from [AutoCommand](#)

- virtual void `on_timeout ()`
- [AutoCommand \\* withTimeout \(double `t\_seconds`\)](#)
- [AutoCommand \\* withCancelCondition \(`Condition` \\*`true\_to\_end`\)](#)

## Additional Inherited Members

## Public Attributes inherited from [AutoCommand](#)

- double `timeout_seconds` = `default_timeout`
- `Condition` \* `true_to_end` = `nullptr`

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double `default_timeout` = 10.0

### 5.83.1 Detailed Description

[AutoCommand](#) that listens to the [Flywheel](#) and waits until it is at its target speed +/- the specified threshold

### 5.83.2 Constructor & Destructor Documentation

#### 5.83.2.1 [WaitUntilUpToSpeedCommand\(\)](#)

```
WaitUntilUpToSpeedCommand::WaitUntilUpToSpeedCommand (
    Flywheel & flywheel,
    int threshold_rpm )
```

Create a [WaitUntilUpToSpeedCommand](#)

#### Parameters

<code>flywheel</code>	the flywheel system we are commanding
<code>threshold_rpm</code>	the threshold over and under the flywheel target RPM that we define to be acceptable

### 5.83.3 Member Function Documentation

#### 5.83.3.1 run()

```
bool WaitUntilUpToSpeedCommand::run ( ) [override], [virtual]
```

Run spin\_manual Overrides run from [AutoCommand](#)

##### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

The documentation for this class was generated from the following files:

- include/utils/command\_structure/flywheel\_commands.h
- src/utils/command\_structure/flywheel\_commands.cpp

# Chapter 6

## File Documentation

### 6.1 robot\_specs.h

```
00001 #pragma once
00002 #include "../core/include/utils/controls/pid.h"
00003 #include "../core/include/utils/controls/feedback_base.h"
00004
00011 typedef struct
00012 {
00013     double robot_radius;
00014
00015     double odom_wheel_diam;
00016     double odom_gear_ratio;
00017     double dist_between_wheels;
00018
00019     double drive_correction_cutoff;
00020
00021     Feedback *drive_feedback;
00022     Feedback *turn_feedback;
00023     PID::pid_config_t correction_pid;
00024
00025 } robot_specs_t;
```

### 6.2 custom\_encoder.h

```
00001 #pragma once
00002 #include "vex.h"
00003
00008 class CustomEncoder : public vex::encoder
00009 {
00010     typedef vex::encoder super;
00011
00012     public:
00018     CustomEncoder(vex::triport::port &port, double ticks_per_rev);
00019
00025     void setRotation(double val, vex::rotationUnits units);
00026
00032     void setPosition(double val, vex::rotationUnits units);
00033
00039     double rotation(vex::rotationUnits units);
00040
00046     double position(vex::rotationUnits units);
00047
00053     double velocity(vex::velocityUnits units);
00054
00055
00056     private:
00057     double tick_scalar;
00058 },
```

## 6.3 flywheel.h

```

00001 #pragma once
00002
00003 #include "../core/include/utils/controls/feedforward.h"
00004 #include "vex.h"
00005 #include "../core/include/robot_specs.h"
00006 #include "../core/include/utils/controls/pid.h"
00007 #include "../core/include/utils/command_structure/auto_command.h"
00008 #include "../core/include/subsystems/screen.h"
00009 #include <atomic>
00010
00018 class Flywheel
00019 {
00020
00021 public:
00022 // CONSTRUCTORS, GETTERS, AND SETTERS
00031 Flywheel(vex::motor_group &motors, Feedback &feedback, FeedForward &helper, const double ratio,
00032 Filter &filt);
00033
00037 double get_target() const;
00038
00042 double getRPM() const;
00043
00047 vex::motor_group &get_motors() const;
00048
00055 void spin_manual(double speed, directionType dir = fwd);
00056
00062 void spin_rpm(double rpm);
00063
00067 void stop();
00068
00073 bool is_on_target()
00074 {
00075     return fb.is_on_target();
00076 }
00077
00082 screen::Page *Page() const;
00083
00089 AutoCommand *SpinRpmCmd(int rpm)
00090 {
00091
00092     return new FunctionCommand([this, rpm]()
00093             {spin_rpm(rpm); return true; });
00094 }
00095
00100 AutoCommand *WaitUntilUpToSpeedCmd()
00101 {
00102     return new WaitUntilCondition(
00103         new FunctionCondition([this]()
00104             { return is_on_target(); }));
00105 }
00106
00107 private:
00108     friend class FlywheelPage;
00109     friend int spinRPMTask(void *wheelPointer);
00110
00111 vex::motor_group &motors;
00112 bool task_running = false;
00113 Feedback &fb;
00114 FeedForward &ffd;
00115 vex::mutex fb_mut;
00116 double ratio;
00117 std::atomic<double> target_rpm;
00118 Task rpm_task;
00119 Filter &avger;
00120
00121 // Functions for internal use only
00126 void set_target(double value);
00130 double measure_RPM();
00131
00138 void spin_raw(double speed, directionType dir = fwd);
00139 };

```

## 6.4 lift.h

```

00001 #pragma once
00002
00003 #include "vex.h"
00004 #include "../core/include/utils/controls/pid.h"
00005 #include <iostream>
00006 #include <map>
00007 #include <atomic>

```

```
00008 #include <vector>
00009
00010 using namespace vex;
00011 using namespace std;
00012
00020 template <typename T>
00021 class Lift
00022 {
00023     public:
00024
00031     struct lift_cfg_t
00032     {
00033         double up_speed, down_speed;
00034         double softstop_up, softstop_down;
00035
00036         PID::pid_config_t lift_pid_cfg;
00037     };
00038
00060     Lift(motor_group &lift_motors, lift_cfg_t &lift_cfg, map<T, double> &setpoint_map, limit
00061     *homming_switch=NULL)
00061     : lift_motors(lift_motors), cfg(lift_cfg), lift_pid(cfg.lift_pid_cfg), setpoint_map(setpoint_map),
00062     homming_switch(homming_switch)
00062     {
00063
00064         is_async = true;
00065         setpoint = 0;
00066
00067         // Create a background task that is constantly updating the lift PID, if requested.
00068         // Set once, and forget.
00069         task t([](void* ptr){
00070             Lift &lift = *((Lift*) ptr);
00071
00072             while(true)
00073             {
00074                 if(lift.get_async())
00075                     lift.hold();
00076
00077                 vexDelay(50);
00078             }
00079
00080             return 0;
00081         }, this);
00082
00083     }
00084
00093     void control_continuous(bool up_ctrl, bool down_ctrl)
00094     {
00095         static timer tmr;
00096
00097         double cur_pos = 0;
00098
00099         // Check if there's a hook for a custom sensor. If not, use the motors.
00100         if(get_sensor == NULL)
00101             cur_pos = lift_motors.position(rev);
00102         else
00103             cur_pos = get_sensor();
00104
00105         if(up_ctrl && cur_pos < cfg.softstop_up)
00106         {
00107             lift_motors.spin(directionType::fwd, cfg.up_speed, volt);
00108             setpoint = cur_pos + .3;
00109
00110             // std::cout << "DEBUG OUT: UP " << setpoint << ", " << tmr.time(sec) << ", " << cfg.down_speed <<
00111             "\n";
00112
00113             // Disable the PID while going UP.
00113             is_async = false;
00114         } else if(down_ctrl && cur_pos > cfg.softstop_down)
00115         {
00116             // Lower the lift slowly, at a rate defined by down_speed
00117             if(setpoint > cfg.softstop_down)
00118                 setpoint = setpoint - (tmr.time(sec) * cfg.down_speed);
00119             // std::cout << "DEBUG OUT: DOWN " << setpoint << ", " << tmr.time(sec) << ", " << cfg.down_speed <<
00120             "\n";
00121             is_async = true;
00122         }
00123
00124         // Hold the lift at the last setpoint
00125         is_async = true;
00126
00127         tmr.reset();
00128     }
00129
00138     void control_manual(bool up_btn, bool down_btn, int volt_up, int volt_down)
00139     {
00140         static bool down_hold = false;
```

```

00141     static bool init = true;
00142
00143     // Allow for setting position while still calling this function
00144     if(init || up_btn || down_btn)
00145     {
00146         init = false;
00147         is_async = false;
00148     }
00149
00150     double rev = lift_motors.position(rotationUnits::rev);
00151
00152     if(rev < cfg.softstop_down && down_btn)
00153         down_hold = true;
00154     else if( !down_btn )
00155         down_hold = false;
00156
00157     if(up_btn && rev < cfg.softstop_up)
00158         lift_motors.spin(directionType::fwd, volt_up, voltageUnits::volt);
00159     else if(down_btn && rev > cfg.softstop_down && !down_hold)
00160         lift_motors.spin(directionType::rev, volt_down, voltageUnits::volt);
00161     else
00162         lift_motors.spin(directionType::fwd, 0, voltageUnits::volt);
00163
00164 }
00165
00177 void control_setpoints(bool up_step, bool down_step, vector<T> pos_list)
00178 {
00179     // Make sure inputs are only processed on the rising edge of the button
00180     static bool up_last = up_step, down_last = down_step;
00181
00182     bool up_rising = up_step && !up_last;
00183     bool down_rising = down_step && !down_last;
00184
00185     up_last = up_step;
00186     down_last = down_step;
00187
00188     static int cur_index = 0;
00189
00190     // Avoid an index overflow. Shouldn't happen unless the user changes pos_list between calls.
00191     if(cur_index >= pos_list.size())
00192         cur_index = pos_list.size() - 1;
00193
00194     // Increment or decrement the index of the list, bringing it up or down.
00195     if(up_rising && cur_index < (pos_list.size() - 1))
00196         cur_index++;
00197     else if(down_rising && cur_index > 0)
00198         cur_index--;
00199
00200     // Set the lift to hold the position in the background with the PID loop
00201     set_position(pos_list[cur_index]);
00202     is_async = true;
00203
00204 }
00205
00214 bool set_position(T pos)
00215 {
00216     this->setpoint = setpoint_map[pos];
00217     is_async = true;
00218
00219     return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
00220 }
00221
00228 bool set_setpoint(double val)
00229 {
00230     this->setpoint = val;
00231     return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
00232 }
00233
00237 double get_setpoint()
00238 {
00239     return this->setpoint;
00240 }
00241
00246 void hold()
00247 {
00248     lift_pid.set_target(setpoint);
00249     // std::cout << "DEBUG OUT: SETPOINT " << setpoint << "\n";
00250
00251     if(get_sensor != NULL)
00252         lift_pid.update(get_sensor());
00253     else
00254         lift_pid.update(lift_motors.position(rev));
00255
00256     // std::cout << "DEBUG OUT: ROTATION " << lift_motors.rotation(rev) << "\n\n";
00257
00258     lift_motors.spin(fwd, lift_pid.get(), volt);
00259 }

```

```

00260
00265     void home()
00266     {
00267         static timer tmr;
00268         tmr.reset();
00269
00270         while(tmr.time(sec) < 3)
00271         {
00272             lift_motors.spin(directionType::rev, 6, volt);
00273
00274             if (homming_switch == NULL && lift_motors.current(currentUnits::amp) > 1.5)
00275                 break;
00276             else if (homming_switch != NULL && homming_switch->pressing())
00277                 break;
00278         }
00279
00280         if(reset_sensor != NULL)
00281             reset_sensor();
00282
00283         lift_motors.resetPosition();
00284         lift_motors.stop();
00285
00286     }
00287
00291     bool get_async()
00292     {
00293         return is_async;
00294     }
00295
00301     void set_async(bool val)
00302     {
00303         this->is_async = val;
00304     }
00305
00315     void set_sensor_function(double (*fn_ptr) (void))
00316     {
00317         this->get_sensor = fn_ptr;
00318     }
00319
00326     void set_sensor_reset(void (*fn_ptr) (void))
00327     {
00328         this->reset_sensor = fn_ptr;
00329     }
00330
00331     private:
00332
00333     motor_group &lift_motors;
00334     lift_cfg_t &cfg;
00335     PID lift_pid;
00336     map<T, double> &setpoint_map;
00337     limit *homming_switch;
00338
00339     atomic<double> setpoint;
00340     atomic<bool> is_async;
00341
00342     double (*get_sensor)(void) = NULL;
00343     void (*reset_sensor)(void) = NULL;
00344
00345
00346 };

```

## 6.5 mecanum\_drive.h

```

00001 #pragma once
00002
00003 #include "vex.h"
00004 #include "../core/include/utils/controls/pid.h"
00005
00006 #ifndef PI
00007 #define PI 3.141592654
00008 #endif
00009
00014 class MecanumDrive
00015 {
00016
00017     public:
00018
00022     struct mecanumdrive_config_t
00023     {
00024         // PID configurations for autonomous driving
00025         PID::pid_config_t drive_pid_conf;
00026         PID::pid_config_t drive_gyro_pid_conf;
00027         PID::pid_config_t turn_pid_conf;

```

```

00028
00029     // Diameter of the mecanum wheels
00030     double drive_wheel_diam;
00031
00032     // Diameter of the perpendicular undriven encoder wheel
00033     double lateral_wheel_diam;
00034
00035     // Width between the center of the left and right wheels
00036     double wheelbase_width;
00037
00038 };
00039
00040 MecanumDrive(vex::motor &left_front, vex::motor &right_front, vex::motor &left_rear, vex::motor
00041     &right_rear,
00042         vex::rotation *lateral_wheel=NULL, vex::inertial *imu=NULL, mecanumdrive_config_t
00043     *config=NULL);
00044
00045     void drive_raw(double direction_deg, double magnitude, double rotation);
00046
00047     void drive(double left_y, double left_x, double right_x, int power=2);
00048
00049     bool auto_drive(double inches, double direction, double speed, bool gyro_correction=true);
00050
00051     bool auto_turn(double degrees, double speed, bool ignore_imu=false);
00052
00053     private:
00054
00055     vex::motor &left_front, &right_front, &left_rear, &right_rear;
00056
00057     mecanumdrive_config_t *config;
00058     vex::rotation *lateral_wheel;
00059     vex::inertial *imu;
00060
00061     PID *drive_pid = NULL;
00062     PID *drive_gyro_pid = NULL;
00063     PID *turn_pid = NULL;
00064
00065     bool init = true;
00066
00067 };
00068 
```

## 6.6 odometry\_3wheel.h

```

00001 #pragma once
00002 #include "../core/include/subsystems/odometry/odometry_base.h"
00003 #include "../core/include/subsystems/tank_drive.h"
00004 #include "../core/include/subsystems/custom_encoder.h"
00005
00032 class Odometry3Wheel : public OdometryBase
00033 {
00034     public:
00035
00040     typedef struct
00041     {
00042         double wheelbase_dist;
00043         double off_axis_center_dist;
00044         double wheel_diam;
00045     } odometry3wheel_cfg_t;
00046
00057     Odometry3Wheel(CustomEncoder &lside_fwd, CustomEncoder &rside_fwd, CustomEncoder &off_axis,
00058         odometry3wheel_cfg_t &cfg, bool is_async=true);
00059
00065     pose_t update() override;
00066
00075     void tune(vex::controller &con, TankDrive &drive);
00076
00077     private:
00078
00091     static pose_t calculate_new_pos(double lside_delta_deg, double rside_delta_deg, double
00092         offax_delta_deg, pose_t old_pos, odometry3wheel_cfg_t cfg);
00093
00094     CustomEncoder &lside_fwd, &rside_fwd, &off_axis;
00095     odometry3wheel_cfg_t &cfg;
00096
00097 }; 
```

## 6.7 odometry\_base.h

```
00001 #pragma once 
```

```

00002
00003 #include "vex.h"
00004 #include "../core/include/utils/geometry.h"
00005 #include "../core/include/robot_specs.h"
00006
00007 #ifndef PI
00008 #define PI 3.141592654
00009 #endif
00010
00011
00012
00025 class OdometryBase
00026 {
00027 public:
00028     OdometryBase(bool is_async);
00035
00040     pose_t get_position(void);
00041
00046     virtual void set_position(const pose_t& newpos=zero_pos);
00047
00052     virtual pose_t update() = 0;
00053
00061     static int background_task(void* ptr);
00062
00068     void end_async();
00069
00076     static double pos_diff(pose_t start_pos, pose_t end_pos);
00077
00084     static double rot_diff(pose_t pos1, pose_t pos2);
00085
00094     static double smallest_angle(double start_deg, double end_deg);
00095
00097     bool end_task = false;
00098
00103     double get_speed();
00104
00109     double get_accel();
00110
00115     double get_angular_speed_deg();
00116
00121     double get_angular_accel_deg();
00122
00126     inline static constexpr pose_t zero_pos = {.x=0.0L, .y=0.0L, .rot=90.0L};
00127
00128 protected:
00132     vex::task *handle;
00133
00137     vex::mutex mut;
00138
00142     pose_t current_pos;
00143
00144     double speed;
00145     double accel;
00146     double ang_speed_deg;
00147     double ang_accel_deg;
00148 };

```

## 6.8 odometry\_tank.h

```

00001 #pragma once
00002
00003 #include "../core/include/subsystems/odometry/odometry_base.h"
00004 #include "../core/include/subsystems/custom_encoder.h"
00005 #include "../core/include/utils/geometry.h"
00006 #include "../core/include/utils/vector2d.h"
00007 #include "../core/include/robot_specs.h"
00008
00009 static int background_task(void* odom_obj);
00010
00011
00018 class OdometryTank : public OdometryBase
00019 {
00020 public:
00029     OdometryTank(vex::motor_group &left_side, vex::motor_group &right_side, robot_specs_t &config,
00030                 vex::inertial *imu=NULL, bool is_async=true);
00030
00040     OdometryTank(CustomEncoder &left_custom_enc, CustomEncoder &right_custom_enc, robot_specs_t
00041 &config, vex::inertial *imu=NULL, bool is_async=true);
00041
00051     OdometryTank(vex::encoder &left_vex_enc, vex::encoder &right_vex_enc, robot_specs_t &config,
00052                 vex::inertial *imu=NULL, bool is_async=true);

```

```

00057     pose_t update() override;
00058
00063     void set_position(const pose_t &newpos=zero_pos) override;
00064
00065
00066
00067 private:
00071     static pose_t calculate_new_pos(robot_specs_t &config, pose_t &stored_info, double lside_diff,
00072     double rside_diff, double angle_deg);
00073     vex::motor_group *left_side, *right_side;
00074     CustomEncoder *left_custom_enc, *right_custom_enc;
00075     vex::encoder *left_vex_enc, *right_vex_enc;
00076     vex::inertial *imu;
00077     robot_specs_t &config;
00078
00079     double rotation_offset = 0;
00080
00081 };

```

## 6.9 screen.h

```

00001 #pragma once
00002 #include "vex.h"
00003 #include <vector>
00004 #include <functional>
00005 #include <map>
00006 #include <cassert>
00007 #include "../core/include/subsystems/odometry/odometry_base.h"
00008 #include "../core/include/utils/graph_drawer.h"
00009 #include "../core/include/utils/controls/pid.h"
00010 #include "../core/include/utils/controls/pidff.h"
00011
00012 namespace screen
00013 {
00015     class Page
00016     {
00017     public:
00026         virtual void update(bool was_pressed, int x, int y);
00034         virtual void draw(vex::brain::lcd &screen, bool first_draw,
00035                         unsigned int frame_number);
00036     };
00037
00039     class SliderWidget
00040     {
00041     public:
00048         SliderWidget(double &val, double low, double high, Rect rect, std::string name) : value(val),
00049             low(low), high(high), rect(rect), name(name) {}
00055         bool update(bool was_pressed, int x, int y);
00057         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
00058
00059     private:
00060         double &value;
00061
00062         double low;
00063         double high;
00064
00065         Rect rect;
00066         std::string name = "";
00067     };
00068
00070     class ButtonWidget
00071     {
00072     public:
00077         ButtonWidget(std::function<void(void)> onpress, Rect rect, std::string name) :
00078             onpress(onpress), rect(rect), name(name) {}
00082         ButtonWidget(void (*onpress)(), Rect rect, std::string name) : onpress(onpress), rect(rect),
00083             name(name) {}
00089         bool update(bool was_pressed, int x, int y);
00091         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
00092
00093     private:
00094         std::function<void(void)> onpress;
00095         Rect rect;
00096         std::string name = "";
00097         bool was_pressed_last = false;
00098     };
00099
00106     void start_screen(vex::brain::lcd &screen, std::vector<Page *> pages, int first_page = 0);
00107
00109     void stop_screen();

```

```
00110
00112     using update_func_t = std::function<void(bool, int, int)>;
00113
00115     using draw_func_t = std::function<void(vex::brain::lcd &screen, bool, unsigned int)>;
00116
00117
00119     class StatsPage : public Page
00120     {
00121     public:
00122         StatsPage(std::map<std::string, vex::motor &> motors);
00123         void update(bool was_pressed, int x, int y) override;
00124         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00125
00126     private:
00127         void draw_motor_stats(const std::string &name, vex::motor &mot, unsigned int frame, int x, int
00128 y, vex::brain::lcd &scr);
00129
00130
00131         std::map<std::string, vex::motor &> motors;
00132         static const int y_start = 0;
00133         static const int per_column = 4;
00134         static const int row_height = 20;
00135         static const int row_width = 200;
00136     };
00137
00138
00139     class OdometryPage : public Page
00140     {
00141     public:
00142         OdometryPage(OdometryBase &odom, double robot_width, double robot_height, bool do_trail);
00143         void update(bool was_pressed, int x, int y) override;
00144         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00145
00146     private:
00147         static const int path_len = 40;
00148         static constexpr char const *field_filename = "vex_field_240p.png";
00149
00150
00151         OdometryBase &odom;
00152         double robot_width;
00153         double robot_height;
00154         uint8_t *buf = nullptr;
00155         int buf_size = 0;
00156         pose_t path[path_len];
00157         int path_index = 0;
00158         bool do_trail;
00159     };
00160
00161
00162     class FunctionPage : public Page
00163     {
00164     public:
00165         FunctionPage(update_func_t update_f, draw_func_t draw_t);
00166         void update(bool was_pressed, int x, int y) override;
00167         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00168
00169     private:
00170         update_func_t update_f;
00171         draw_func_t draw_f;
00172     };
00173
00174
00175     class PIDPage : public Page
00176     {
00177     public:
00178         PIDPage(PID &pid, std::string name, std::function<void(void)> onchange = []() {});
00179         PIDPage(PIDFF &pidff, std::string name, std::function<void(void)> onchange = []() {});
00180
00181         void update(bool was_pressed, int x, int y) override;
00182         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00183
00184     private:
00185         void zero_d_f() { cfg.d = 0; }
00186         void zero_i_f() { cfg.i = 0; }
00187
00188
00189         PID::pid_config_t &cfg;
00190         PID &pid;
00191         const std::string name;
00192         std::function<void(void)> onchange;
00193
00194         SliderWidget p_slider;
00195         SliderWidget i_slider;
00196         SliderWidget d_slider;
00197         ButtonWidget zero_i;
00198         ButtonWidget zero_d;
00199
00200
00201         GraphDrawer graph;
00202     };
00203
00204
00205 }
```

## 6.10 tank\_drive.h

```

00001 #pragma once
00002
00003 #ifndef PI
00004 #define PI 3.141592654
00005 #endif
00006
00007 #include "vex.h"
00008 #include "../core/include/subsystems/odometry/odometry_tank.h"
00009 #include "../core/include/utils/controls/pid.h"
00010 #include "../core/include/utils/controls/feedback_base.h"
00011 #include "../core/include/robot_specs.h"
00012 #include "../core/include/utils/pure_pursuit.h"
00013 #include "../core/include/utils/command_structure/auto_command.h"
00014 #include <vector>
00015
00016 using namespace vex;
00017
00018 class TankDrive
00019 {
00020 public:
00021     TankDrive(motor_group &left_motors, motor_group &right_motors, robot_specs_t &config, OdometryBase
00022     *odom = NULL);
00023
00024     AutoCommand *DriveToPointCmd(point_t pt, vex::directionType dir = vex::forward, double max_speed =
00025     1.0, double end_speed = 0.0);
00026     AutoCommand *DriveToPointCmd(Feedback &fb, point_t pt, vex::directionType dir = vex::forward, double
00027     max_speed = 1.0, double end_speed = 0.0);
00028     AutoCommand *DriveForwardCmd(double dist, vex::directionType dir = vex::forward, double max_speed =
00029     1.0, double end_speed = 0.0);
00030     AutoCommand *DriveForwardCmd(Feedback &fb, double dist, vex::directionType dir = vex::forward,
00031     double max_speed = 1.0, double end_speed = 0.0);
00032
00033     AutoCommand *TurnToHeadingCmd(double heading, double max_speed = 1.0, double end_speed = 0.0);
00034     AutoCommand *TurnToHeadingCmd(Feedback &fb, double heading, double max_speed = 1.0, double end_speed
00035     = 0.0);
00036
00037     AutoCommand *TurnDegreesCmd(double degrees, double max_speed = 1.0, double start_speed = 0.0);
00038     AutoCommand *TurnDegreesCmd(Feedback &fb, double degrees, double max_speed = 1.0, double end_speed =
00039     0.0);
00040
00041     AutoCommand *PurePursuitCmd(PurePursuit::Path path, directionType dir, double max_speed=1, double
00042     end_speed=0);
00043     AutoCommand *PurePursuitCmd(Feedback &feedback, PurePursuit::Path path, directionType dir, double
00044     max_speed=1, double end_speed=0);
00045
00046     void stop();
00047
00048     void drive_tank(double left, double right, int power=1);
00049
00050     void drive_arcade(double forward_back, double left_right, int power = 1);
00051
00052     bool drive_forward(double inches, directionType dir, Feedback &feedback, double max_speed = 1,
00053     double end_speed = 0);
00054
00055     bool drive_forward(double inches, directionType dir, double max_speed = 1, double end_speed = 0);
00056
00057     bool turn_degrees(double degrees, Feedback &feedback, double max_speed = 1, double end_speed = 0);
00058
00059     bool turn_degrees(double degrees, double max_speed = 1, double end_speed = 0);
00060
00061     bool drive_to_point(double x, double y, vex::directionType dir, Feedback &feedback, double max_speed
00062     = 1, double end_speed = 0);
00063
00064     bool drive_to_point(double x, double y, vex::directionType dir, double max_speed = 1, double
00065     end_speed = 0);
00066
00067     bool turn_to_heading(double heading_deg, Feedback &feedback, double max_speed = 1, double end_speed
00068     = 0);
00069     bool turn_to_heading(double heading_deg, double max_speed = 1, double end_speed = 0);
00070
00071     void reset_auto();
00072
00073     static double modify_inputs(double input, int power = 2);
00074
00075     bool pure_pursuit(PurePursuit::Path path, directionType dir, Feedback &feedback, double max_speed=1,
00076     double end_speed=0);
00077
00078     bool pure_pursuit(PurePursuit::Path path, directionType dir, double max_speed=1, double
00079     end_speed=0);
00080
00081 private:
00082     motor_group &left_motors;
00083     motor_group &right_motors;
00084
00085
00086
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135
00136
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184
00185
00186
00187
00188
00189
00190
00191
00192
00193
00194
00195
00196
00197
00198
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221

```

```

00222     PID correction_pid;
00223     Feedback *drive_default_feedback = NULL;
00224     Feedback *turn_default_feedback = NULL;
00225
00226     OdometryBase *odometry;
00227
00228     robot_specs_t &config;
00229
00230     bool func_initialized = false;
00231     bool is_pure_pursuit = false;
00232 };

```

## 6.11 auto\_chooser.h

```

00001 #pragma once
00002 #include "vex.h"
00003 #include <string>
00004 #include <vector>
00005 #include "../core/include/subsystems/screen.h"
00006 #include "../core/include/utils/geometry.h"
00007
00016 class AutoChooser : public screen::Page
00017 {
00018 public:
00024     AutoChooser(std::vector<std::string> paths, size_t def = 0);
00025
00026     void update(bool was_pressed, int x, int y);
00027     void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
00028
00033     size_t get_choice();
00034
00035 protected:
00039     struct entry_t
00040     {
00041         Rect rect;
00042         std::string name;
00043     };
00044
00045     static const size_t width = 380;
00046     static const size_t height = 220;
00047
00048     size_t choice;
00049     std::vector<entry_t> list ;
00050 };

```

## 6.12 auto\_command.h

```

00001
00007 #pragma once
00008
00009 #include "vex.h"
00010 #include <functional>
00011 #include <vector>
00012 #include <queue>
00013 #include <atomic>
00014
00015
00025 class Condition
00026 {
00027 public:
00028     Condition *Or(Condition *b);
00029     Condition *And(Condition *b);
00030     virtual bool test() = 0;
00031 };
00032
00033
00034 class AutoCommand
00035 {
00036 public:
00037     static constexpr double default_timeout = 10.0;
00043     virtual bool run() { return true; }
00047     virtual void on_timeout() {}
00048     AutoCommand *withTimeout(double t_seconds)
00049     {
00050         if (this->timeout_seconds < 0)
00051         {
00052             // should never be timed out
00053             return this;
00054         }

```

```

00055     this->timeout_seconds = t_seconds;
00056     return this;
00057 }
00058 AutoCommand *withCancelCondition(Condition *true_to_end) {
00059     this->true_to_end = true_to_end;
00060     return this;
00061 }
00062 double timeout_seconds = default_timeout;
00063 Condition *true_to_end = nullptr;
00064 };
00065
00066 class FunctionCommand : public AutoCommand
00067 {
00068 public:
00069     FunctionCommand(std::function<bool(void)> f) : f(f) {}
00070     bool run()
00071     {
00072         return f();
00073     }
00074
00075 private:
00076     std::function<bool(void)> f;
00077 };
00078
00079 // Times tested 3
00080 // Test 1 -> false
00081 // Test 2 -> false
00082 // Test 3 -> true
00083 // Returns false until the Nth time that it is called
00084 // This is pretty much only good for implementing RepeatUntil
00085 class TimesTestedCondition : public Condition
00086 {
00087 public:
00088     TimesTestedCondition(size_t N) : max(N) {}
00089     bool test() override
00090     {
00091         count++;
00092         if (count >= max)
00093             return true;
00094         return false;
00095     }
00096
00097 private:
00098     size_t count = 0;
00099     size_t max;
00100 };
00101
00102 class FunctionCondition : public Condition
00103 {
00104 public:
00105     FunctionCondition(
00106         std::function<bool(void)> cond, std::function<void(void)> timeout = []() {}) : cond(cond),
00107         timeout(timeout)
00108     {
00109     }
00110     bool test() override;
00111
00112 private:
00113     std::function<bool(void)> cond;
00114     std::function<void(void)> timeout;
00115 };
00116
00117 class IfTimePassed : public Condition
00118 {
00119 public:
00120     IfTimePassed(double time_s);
00121     bool test() override;
00122
00123 private:
00124     double time_s;
00125     vex::timer tmr;
00126 };
00127
00128 class WaitUntilCondition : public AutoCommand
00129 {
00130 public:
00131     WaitUntilCondition(Condition *cond) : cond(cond) {}
00132     bool run() override
00133     {
00134         return cond->test();
00135     }
00136
00137 private:
00138     Condition *cond;
00139 };
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156

```

```

00157
00160
00163 class InOrder : public AutoCommand
00164 {
00165 public:
00166     InOrder(const InOrder &other) = default;
00167     InOrder(std::queue<AutoCommand *> cmdbs);
00168     InOrder(std::initializer_list<AutoCommand *> cmdbs);
00169     bool run() override;
00170     void on_timeout() override;
00171
00172 private:
00173     AutoCommand *current_command = nullptr;
00174     std::queue<AutoCommand *> cmdbs;
00175     vex::timer tmr;
00176 };
00177
00178 class Parallel : public AutoCommand
00179 {
00180 public:
00181     Parallel(std::initializer_list<AutoCommand *> cmdbs);
00182     bool run() override;
00183     void on_timeout() override;
00184
00185 private:
00186     std::vector<AutoCommand *> cmdbs;
00187     std::vector<vex::task *> runners;
00188 };
00189
00190 class Branch : public AutoCommand
00191 {
00192 public:
00193     Branch(Condition *cond, AutoCommand *false_choice, AutoCommand *true_choice);
00194     ~Branch();
00195     bool run() override;
00196     void on_timeout() override;
00197
00198 private:
00199     AutoCommand *false_choice;
00200     AutoCommand *true_choice;
00201     Condition *cond;
00202     bool choice = false;
00203     bool chosen = false;
00204     vex::timer tmr;
00205 };
00206
00207 class Async : public AutoCommand
00208 {
00209 public:
00210     Async(AutoCommand *cmd) : cmd(cmd) {}
00211     bool run() override;
00212
00213 private:
00214     AutoCommand *cmd = nullptr;
00215 };
00216
00217 class RepeatUntil : public AutoCommand
00218 {
00219 public:
00220     RepeatUntil(InOrder cmdbs, size_t repeats);
00221     RepeatUntil(InOrder cmdbs, Condition *true_to_end);
00222     bool run() override;
00223     void on_timeout() override;
00224
00225 private:
00226     const InOrder cmdbs;
00227     InOrder *working_cmds;
00228     Condition *cond;
00229 };
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243 };

```

## 6.13 basic\_command.h

```

00001
00014 #pragma once
00015
00016 #include "../core/include/utils/command_structure/auto_command.h"
00017
00018 //Basic Motor Classes-----
00019
00024 class BasicSpinCommand : public AutoCommand {
00025     public:
00026         //Enumurator for the type of power setting in the motor

```

```

00028     enum type {percent,voltage,veocity};
00029
00038     BasicSpinCommand(vex::motor &motor, vex::directionType dir, BasicSpinCommand::type setting,
00039                         double power);
00046     bool run() override;
00047
00048     private:
00049
00050     vex::motor &motor;
00051
00052     type setting;
00053
00054     vex::directionType dir;
00055
00056     double power;
00057 };
00062 class BasicStopCommand : public AutoCommand{
00063     public:
00064
00071     BasicStopCommand(vex::motor &motor, vex::brakeType setting);
00072
00079     bool run() override;
00080
00081     private:
00082
00083     vex::motor &motor;
00084
00085     vex::brakeType setting;
00086 };
00087
00088 //Basic Solenoid Commands-----
00089
00094 class BasicSolenoidSet : public AutoCommand{
00095     public:
00096
00103     BasicSolenoidSet(vex::pneumatics &solenoid, bool setting);
00104
00111     bool run() override;
00112
00113     private:
00114
00115     vex::pneumatics &solenoid;
00116
00117     bool setting;
00118 };

```

## 6.14 command\_controller.h

```

00001
00010 #pragma once
00011 #include <vector>
00012 #include <queue>
00013 #include "../core/include/utils/command_structure/auto_command.h"
00014
00015 class CommandController
00016 {
00017     public:
00019     [[deprecated("Use list constructor instead.")]] CommandController() : command_queue({}) {}
00020
00023     CommandController(std::initializer_list<AutoCommand *> cmd) : command_queue(cmd) {}
00029     [[deprecated("Use list constructor instead. If you need to make a decision before adding new
00030     commands, use Branch (https://github.com/RIT-VEX-U/Core/wiki/3-%7C-Utilites#commandcontroller)")]
00031     void add(std::vector<AutoCommand *> cmd);
00030     void add(AutoCommand *cmd, double timeout_seconds = 10.0);
00031
00042     [[deprecated("Use list constructor instead. If you need to make a decision before adding new
00043     commands, use Branch (https://github.com/RIT-VEX-U/Core/wiki/3-%7C-Utilites#commandcontroller)")]
00044     void
00043     add(std::vector<AutoCommand *> cmd, double timeout_sec);
00050     void add_delay(int ms);
00051
00054     void add_cancel_func(std::function<bool(void)> true_if_cancel);
00055
00060     void run();
00061
00067     bool last_command_timed_out();
00068
00069     private:
00070     std::queue<AutoCommand *> command_queue;
00071     bool command_timed_out = false;
00072     std::function<bool()> should_cancel = [](){}
00073     { return false; };
00074 };

```

## 6.15 delay\_command.h

```

00001
00008 #pragma once
00009
00010 #include "../core/include/utils/command_structure/auto_command.h"
00011
00012 class DelayCommand: public AutoCommand {
00013     public:
00014         DelayCommand(int ms): ms(ms) {}
00015
00016         bool run() override {
00017             vexDelay(ms);
00018             return true;
00019         }
00020
00021     private:
00022         // amount of milliseconds to wait
00023         int ms;
00024     };

```

## 6.16 drive\_commands.h

```

00001
00019 #pragma once
00020
00021 #include "vex.h"
00022 #include "../core/include/utils/geometry.h"
00023 #include "../core/include/utils/command_structure/auto_command.h"
00024 #include "../core/include/subsystems/tank_drive.h"
00025
00026 using namespace vex;
00027
00028
00029 // ===== DRIVING =====
00030
00036 class DriveForwardCommand: public AutoCommand
00037 {
00038     public:
00039         DriveForwardCommand(TankDrive &drive_sys, Feedback &feedback, double inches, directionType dir,
00040         double max_speed=1, double end_speed=0);
00041
00046         bool run() override;
00047         void on_timeout() override;
00048
00052     private:
00053         // drive system to run the function on
00054         TankDrive &drive_sys;
00055
00056         // feedback controller to use
00057         Feedback &feedback;
00058
00059         // parameters for drive_forward
00060         double inches;
00061         directionType dir;
00062         double max_speed;
00063         double end_speed;
00064     };
00065
00070 class TurnDegreesCommand: public AutoCommand
00071 {
00072     public:
00073         TurnDegreesCommand(TankDrive &drive_sys, Feedback &feedback, double degrees, double max_speed = 1,
00074         double end_speed = 0);
00075
00080         bool run() override;
00081         void on_timeout() override;
00082
00086
00087     private:
00088         // drive system to run the function on
00089         TankDrive &drive_sys;
00090
00091         // feedback controller to use
00092         Feedback &feedback;
00093
00094         // parameters for turn_degrees
00095         double degrees;
00096         double max_speed;
00097         double end_speed;
00098     };
00099
00104 class DriveToPointCommand: public AutoCommand

```

```

00105 {
00106     public:
00107         DriveToPointCommand(TankDrive &drive_sys, Feedback &feedback, double x, double y, directionType
00108             dir, double max_speed = 1, double end_speed = 0);
00109         DriveToPointCommand(TankDrive &drive_sys, Feedback &feedback, point_t point, directionType dir,
00110             double max_speed=1, double end_speed = 0);
00111
00112     bool run() override;
00113
00114     private:
00115         // drive system to run the function on
00116         TankDrive &drive_sys;
00117
00118     void on_timeout() override;
00119
00120
00121     // feedback controller to use
00122     Feedback &feedback;
00123
00124     // parameters for drive_to_point
00125     double x;
00126     double y;
00127     directionType dir;
00128     double max_speed;
00129     double end_speed;
00130
00131 };
00132
00133 class TurnToHeadingCommand: public AutoCommand
00134 {
00135     public:
00136         TurnToHeadingCommand(TankDrive &drive_sys, Feedback &feedback, double heading_deg, double speed =
00137             1, double end_speed = 0);
00138
00139     bool run() override;
00140     void on_timeout() override;
00141
00142     private:
00143         // drive system to run the function on
00144         TankDrive &drive_sys;
00145
00146         // feedback controller to use
00147         Feedback &feedback;
00148
00149         // parameters for turn_to_heading
00150         double heading_deg;
00151         double max_speed;
00152         double end_speed;
00153
00154 };
00155
00156 class PurePursuitCommand: public AutoCommand
00157 {
00158     public:
00159         PurePursuitCommand(TankDrive &drive_sys, Feedback &feedback, PurePursuit::Path path, directionType
00160             dir, double max_speed=1, double end_speed=0);
00161
00162     bool run() override;
00163
00164     void on_timeout() override;
00165
00166     private:
00167         TankDrive &drive_sys;
00168         PurePursuit::Path path;
00169         directionType dir;
00170         Feedback &feedback;
00171         double max_speed;
00172         double end_speed;
00173
00174 };
00175
00176 class DriveStopCommand: public AutoCommand
00177 {
00178     public:
00179         DriveStopCommand(TankDrive &drive_sys);
00180
00181     bool run() override;
00182     void on_timeout() override;
00183
00184     private:
00185         // drive system to run the function on
00186         TankDrive &drive_sys;
00187
00188 };
00189
00190
00191 // ===== ODOMETRY =====
00192
00193
00194
00195
00196
00197
00198
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233 // ===== ODOMETRY =====
00234

```

```

00239 class OdomSetPosition: public AutoCommand
00240 {
00241     public:
00242         OdomSetPosition(OdometryBase &odom, const pose_t &newpos=OdometryBase::zero_pos);
00243
00244     bool run() override;
00245
00246     private:
00247         // drive system with an odometry config
00248         OdometryBase &odom;
00249         pose_t newpos;
00250 };
00251

```

## 6.17 flywheel\_commands.h

```

00001
00007 #pragma once
00008
00009 #include "../core/include/subsystems/flywheel.h"
00010 #include "../core/include/utils/command_structure/auto_command.h"
00011
00017 class SpinRPMCommand: public AutoCommand {
00018     public:
00019         SpinRPMCommand(Flywheel &flywheel, int rpm);
00020
00021     bool run() override;
00022
00023     private:
00024         // Flywheel instance to run the function on
00025         Flywheel &flywheel;
00026
00027         // parameters for spin_rpm
00028         int rpm;
00029 };
00030
00045 class WaitUntilUpToSpeedCommand: public AutoCommand {
00046     public:
00047         WaitUntilUpToSpeedCommand(Flywheel &flywheel, int threshold_rpm);
00048
00049     bool run() override;
00050
00051     private:
00052         // Flywheel instance to run the function on
00053         Flywheel &flywheel;
00054
00055         // if the actual speed is equal to the desired speed +/- this value, we are ready to fire
00056         int threshold_rpm;
00057 };
00058
00074 class FlywheelStopCommand: public AutoCommand {
00075     public:
00076         FlywheelStopCommand(Flywheel &flywheel);
00077
00078     bool run() override;
00079
00080     private:
00081         // Flywheel instance to run the function on
00082         Flywheel &flywheel;
00083 };
00084
00099 class FlywheelStopMotorsCommand: public AutoCommand {
00100     public:
00101         FlywheelStopMotorsCommand(Flywheel &flywheel);
00102
00103     bool run() override;
00104
00105     private:
00106         // Flywheel instance to run the function on
00107         Flywheel &flywheel;
00108 };
00109
00124 class FlywheelStopNonTasksCommand: public AutoCommand {
00125     public:
00126         FlywheelStopNonTasksCommand(Flywheel &flywheel);
00127
00128     bool run() override;
00129
00130     private:
00131         // Flywheel instance to run the function on
00132         Flywheel &flywheel;
00133 };
00134

```

## 6.18 bang\_bang.h

```

00001 #include "../core/include/utils/controls/feedback_base.h"
00002
00003 class BangBang : public Feedback
00004 {
00005
00006 public:
00007     BangBang(double threshold, double low, double high);
00016     void init(double start_pt, double set_pt, double start_vel [[maybe_unused]] = 0.0, double end_vel
00016     [[maybe_unused]] = 0.0) override;
00017
00024     double update(double val) override;
00025
00029     double get() override;
00030
00037     void set_limits(double lower, double upper) override;
00038
00042     bool is_on_target() override;
00043
00044 private:
00045     double setpt;
00046     double sensor_val;
00047     double lower_bound, upper_bound;
00048     double last_output;
00049     double threshold;
00050 };

```

## 6.19 feedback\_base.h

```

00001 #pragma once
00002
00010 class Feedback
00011 {
00012 public:
00021     virtual void init(double start_pt, double set_pt, double start_vel = 0.0, double end_vel = 0.0) =
0;
00022
00029     virtual double update(double val) = 0;
00030
00034     virtual double get() = 0;
00035
00042     virtual void set_limits(double lower, double upper) = 0;
00043
00047     virtual bool is_on_target() = 0;
00048
00049
00050 };

```

## 6.20 feedforward.h

```

00001 #pragma once
00002
00003 #include <math.h>
00004 #include <vector>
00005 #include "../core/include/utils/math_util.h"
00006 #include "../core/include/utils/moving_average.h"
00007 #include "vex.h"
00008
00029 class FeedForward
00030 {
00031     public:
00032
00041     typedef struct
00042     {
00043         double kS;
00044         double KV;
00045         double kA;
00046         double KG;
00047     } ff_config_t;
00048
00049
00054     FeedForward(ff_config_t &cfg) : cfg(cfg) {}
00055
00066     double calculate(double v, double a, double pid_ref=0.0)
00067     {
00068         double ks_sign = 0;
00069         if(v != 0)
00070             ks_sign = sign(v);

```

```

00071     else if(pid_ref != 0)
00072         ks_sign = sign(pid_ref);
00073
00074     return (cfg.KS * ks_sign) + (cfg.KV * v) + (cfg.KA * a) + cfg.KG;
00075 }
00076
00077 private:
00078
00079     ff_config_t &cfg;
00080
00081 };
00082
00083
00091 FeedForward::ff_config_t tune_feedforward(vex::motor_group &motor, double pct, double duration);

```

## 6.21 motion\_controller.h

```

00001 #pragma once
00002 #include "../core/include/utils/controls/pid.h"
00003 #include "../core/include/utils/controls/feedforward.h"
00004 #include "../core/include/utils/controls/trapezoid_profile.h"
00005 #include "../core/include/utils/controls/feedback_base.h"
00006 #include "../core/include/subsystems/tank_drive.h"
00007 #include "vex.h"
00008
00025 class MotionController : public Feedback
00026 {
00027     public:
00028
00034     typedef struct
00035     {
00036         double max_v;
00037         double accel;
00038         PID::pid_config_t pid_cfg;
00039         FeedForward::ff_config_t ff_cfg;
00040     } m_profile_cfg_t;
00041
00051     MotionController(m_profile_cfg_t &config);
00052
00057     void init(double start_pt, double end_pt, double start_vel, double end_vel) override;
00058
00065     double update(double sensor_val) override;
00066
00070     double get() override;
00071
00079     void set_limits(double lower, double upper) override;
00080
00085     bool is_on_target() override;
00086
00090     motion_t get_motion();
00091
00110     static FeedForward::ff_config_t tune_feedforward(TankDrive &drive, OdometryTank &odometry, double
00111     pct=0.6, double duration=2);
00112
00113     private:
00114     m_profile_cfg_t config;
00115
00116     PID pid;
00117     FeedForward ff;
00118     TrapezoidProfile profile;
00119
00120     double current_pos;
00121     double end_pt;
00122
00123     double lower_limit = 0, upper_limit = 0;
00124     double out = 0;
00125     motion_t cur_motion;
00126
00127     vex::timer tmr;
00128
00129 };

```

## 6.22 pid.h

```

00001 #pragma once
00002
00003 #include <cmath>
00004 #include "vex.h"

```

```

00005 #include "../core/include/utils/controls/feedback_base.h"
00006
00007 using namespace vex;
00008
00023 class PID : public Feedback
00024 {
00025 public:
00029     enum ERROR_TYPE{
00030         LINEAR,
00031         ANGULAR // assumes degrees
00032     };
00040     struct pid_config_t
00041     {
00042         double p;
00043         double i;
00044         double d;
00045         double deadband;
00046         double on_target_time;
00047         ERROR_TYPE error_method;
00048     };
00049
00050
00051
00056     PID(pid_config_t &config);
00057
00058
00069     void init(double start_pt, double set_pt, double start_vel = 0, double end_vel = 0) override;
00070
00077     double update(double sensor_val) override;
00078
00079
00084     double get_sensor_val();
00085
00086
00091     double get() override;
00092
00099     void set_limits(double lower, double upper) override;
00100
00105     bool is_on_target() override;
00106
00110     void reset();
00111
00116     double get_error();
00117
00122     double get_target();
00123
00128     void set_target(double target);
00129
00130
00131     pid_config_t &config;
00132
00133 private:
00134
00135
00136     double last_error = 0;
00137     double accum_error = 0;
00138
00139     double last_time = 0;
00140     double on_target_last_time = 0;
00141
00142     double lower_limit = 0;
00143     double upper_limit = 0;
00144
00145     double target = 0;
00146     double target_vel = 0;
00147     double sensor_val = 0;
00148     double out = 0;
00149
00150     bool is_checking_on_target = false;
00151
00152     timer pid_timer;
00153 };

```

## 6.23 pidff.h

```

00001 #pragma once
00002 #include "../core/include/utils/controls/feedback_base.h"
00003 #include "../core/include/utils/controls/pid.h"
00004 #include "../core/include/utils/controls/feedforward.h"
00005
00006 class PIDFF : public Feedback
00007 {
00008     public:

```

```

00009
00010     PIDFF(PID::pid_config_t &pid_cfg, FeedForward::ff_config_t &ff_cfg);
00011
00020     void init(double start_pt, double set_pt, double start_vel, double end_vel) override;
00021
00026     void set_target(double set_pt);
00027
00035     double update(double val) override;
00036
00045     double update(double val, double vel_setpt, double a_setpt=0);
00046
00050     double get() override;
00051
00058     void set_limits(double lower, double upper) override;
00059
00063     bool is_on_target() override;
00064
00065     PID pid;
00066
00067
00068     private:
00069
00070     FeedForward::ff_config_t &ff_cfg;
00071
00072     FeedForward ff;
00073
00074     double out;
00075     double lower_lim, upper_lim;
00076
00077 };

```

## 6.24 take\_back\_half.h

```

00001 #pragma once
00002 #include "../core/include/utils/controls/feedback_base.h"
00003
00006 class TakeBackHalf : public Feedback
00007 {
00008
00009     public:
00010     TakeBackHalf(double TBH_gain, double first_cross_split, double on_target_threshold);
00019     void init(double start_pt, double set_pt, double, double);
00026     double update(double val) override;
00027
00031     double get() override;
00032
00039     void set_limits(double lower, double upper) override;
00040
00044     bool is_on_target() override;
00045
00046     double TBH_gain;
00047     double first_cross_split;
00048
00049     private:
00049     double on_target_threshold;
00050
00051     double target = 0.0;
00052
00053     bool first_cross = true;
00054     double tbh = 0.0;
00055     double prev_error = 0.0;
00056
00057     double output = 0.0;
00058     double lower = 0.0, upper = 0.0;
00059 };

```

## 6.25 trapezoid\_profile.h

```

00001 #pragma once
00002
00003 const int MAX_TRAPEZOID_PROFILE_SEGMENTS = 4;
00004
00008 typedef struct
00009 {
00010     double pos;
00011     double vel;
00012     double accel;
00013
00014 } motion_t;
00015

```

```

00019 typedef struct
00020 {
00021     double pos_after;
00022     double vel_after;
00023     double accel;
00024     double duration;
00025 } trapezoid_profile_segment_t;
00026
00060 class TrapezoidProfile
00061 {
00062     public:
00063
00070     TrapezoidProfile(double max_v, double accel);
00071
00079     motion_t calculate(double time_s, double pos_s);
00080
00087     motion_t calculate_time_based(double time_s);
00088
00095     void set_endpts(double start, double end);
00096
00103     void set_vel_endpts(double start, double end);
00104
00110     void set_accel(double accel);
00111
00117     void set_max_v(double max_v);
00118
00124     double get_movement_time();
00125
00126     private:
00127     double si, sf;
00128     double vi, vf;
00129     double max_v;
00130     double accel;
00131
00132     trapezoid_profile_segment_t segments[MAX_TRAPEZOID_PROFILE_SEGMENTS];
00133     int num_acceleration_phases;
00134
00135     bool precalculated;
00136
00142     bool precalculate();
00143
00152     trapezoid_profile_segment_t calculate_kinetic_motion(double si, double vi, double v_target);
00153
00161     trapezoid_profile_segment_t calculate_next_segment(double s, double v);
00162 };

```

## 6.26 generic\_auto.h

```

00001 #pragma once
00002
00003 #include <queue>
00004 #include <map>
00005 #include "vex.h"
00006 #include <functional>
00007
00008 typedef std::function<bool(void)> state_ptr;
00009
00014 class GenericAuto
00015 {
00016     public:
00017
00031     [[deprecated("Use CommandController instead.")]]
00032     bool run(bool blocking);
00033
00038     [[deprecated("Use CommandController instead.")]]
00039     void add(state_ptr new_state);
00040
00045     [[deprecated("Use CommandController instead.")]]
00046     void add_async(state_ptr async_state);
00047
00052     [[deprecated("Use CommandController instead.")]]
00053     void add_delay(int ms);
00054
00055     private:
00056
00057     std::queue<state_ptr> state_list;
00058
00059 };

```

## 6.27 geometry.h

```

00001 #pragma once
00002 #include <cmath>
00003
00004 struct point_t
00005 {
00006     double x;
00007     double y;
00008
00009     double dist(const point_t other) const
00010     {
00011         return std::sqrt(std::pow(this->x - other.x, 2) + pow(this->y - other.y, 2));
00012     }
00013
00014     point_t operator+(const point_t &other) const
00015     {
00016         point_t p{
00017             .x = this->x + other.x,
00018             .y = this->y + other.y};
00019         return p;
00020     }
00021
00022     point_t operator-(const point_t &other) const
00023     {
00024         point_t p{
00025             .x = this->x - other.x,
00026             .y = this->y - other.y};
00027         return p;
00028     }
00029
00030     point_t operator*(double s) const
00031     {
00032         return {x * s, y * s};
00033     }
00034
00035     point_t operator/(double s) const
00036     {
00037         return {x / s, y / s};
00038     }
00039
00040     point_t operator-() const
00041     {
00042         return {-x, -y};
00043     }
00044
00045     point_t operator+() const
00046     {
00047         return {x, y};
00048     }
00049
00050     bool operator==(const point_t &rhs)
00051     {
00052         return x == rhs.x && y == rhs.y;
00053     }
00054 };
00055
00056 struct pose_t
00057 {
00058     double x;
00059     double y;
00060     double rot;
00061
00062     point_t get_point()
00063     {
00064         return point_t{.x = x, .y = y};
00065     }
00066
00067     static Rect from_min_and_size(point_t min, point_t size)
00068     {
00069         return {min, min+size};
00070     }
00071
00072     point_t dimensions() const
00073     {
00074         return max - min;
00075     }
00076
00077     point_t center() const
00078     {
00079         return (min + max)/2;
00080     }
00081
00082     double width() const
00083     {
00084         return max.x - min.x;
00085     }
00086
00087     double height() const
00088     {
00089         return max.y - min.y;
00090     }
00091
00092 }
```

```

00107     }
00108     bool contains(point_t p) const
00109     {
00110         bool xin = p.x > min.x && p.x < max.x;
00111         bool yin = p.y > min.y && p.y < max.y;
00112         return xin && yin;
00113     }
00114
00115 };
00116
00117 struct Mat2
00118 {
00119     double X11, X12;
00120     double X21, X22;
00121     point_t operator*(const point_t p) const
00122     {
00123         double outx = p.x * X11 + p.y * X12;
00124         double outy = p.x * X21 + p.y * X22;
00125         return {outx, outy};
00126     }
00127
00128     static Mat2 FromRotationDegrees(double degrees)
00129     {
00130         double rad = degrees * (M_PI / 180.0);
00131         double c = cos(rad);
00132         double s = sin(rad);
00133         return {c, -s, s, c};
00134     }
00135 };

```

## 6.28 graph\_drawer.h

```

00001 #pragma once
00002
00003 #include <string>
00004 #include <stdio.h>
00005 #include <vector>
00006 #include <cmath>
00007 #include "vex.h"
00008 #include "../core/include/utils/geometry.h"
00009 #include "../core/include/utils/vector2d.h"
00010
00011 class GraphDrawer
00012 {
00013 public:
00020     GraphDrawer(int num_samples, double lower_bound, double upper_bound, std::vector<vex::color> colors,
00021     size_t num_series = 1);
00025     void add_samples(std::vector<point_t> sample);
00026
00031     void add_samples(std::vector<double> sample);
00032
00040     void draw(vex::brain::lcd &screen, int x, int y, int width, int height);
00041
00042 private:
00043     std::vector<std::vector<point_t>> series;
00044     int sample_index = 0;
00045     std::vector<vex::color> cols;
00046     vex::color bgcol = vex::transparent;
00047     bool border;
00048     double upper;
00049     double lower;
00050     bool auto_fit = false;
00051 };

```

## 6.29 logger.h

```

00001 #pragma once
00002
00003 #include <cstdarg>
00004 #include <cstdio>
00005 #include <string>
00006 #include "vex.h"
00007
00009 enum LogLevel
00010 {
00011     DEBUG,
00012     NOTICE,
00013     WARNING,
00014     ERROR,

```

```

00015     CRITICAL,
00016     TIME
00017 };
00018
00019 class Logger
00020 {
00021 {
00022 private:
00023     const std::string filename;
00024     vex::brain::sdcard sd;
00025     void write_level(LogLevel l);
00026
00027 public:
00028     static constexpr int MAX_FORMAT_LEN = 512;
00029     explicit Logger(const std::string &filename);
00030
00031     Logger(const Logger &l) = delete;
00032     Logger &operator=(const Logger &l) = delete;
00033
00034     void Log(const std::string &s);
00035     void Log(LogLevel level, const std::string &s);
00036
00037     void Logln(const std::string &s);
00038     void Logln(LogLevel level, const std::string &s);
00039
00040     void Logf(const char *fmt, ...);
00041     void Logf(LogLevel level, const char *fmt, ...);
00042 };
00043

```

## 6.30 math\_util.h

```

00001 #pragma once
00002 #include <vector>
00003 #include "math.h"
00004 #include "vex.h"
00005 #include "../core/include/utils/geometry.h"
00006
00007
00015 double clamp(double value, double low, double high);
00016
00023 double lerp(double a, double b, double t);
00030 double sign(double x);
00031
00032 double wrap_angle_deg(double input);
00033 double wrap_angle_rad(double input);
00034
00035 /*
00036 Calculates the variance of a set of numbers (needed for linear regression)
00037 https://en.wikipedia.org/wiki/Variance
00038 @param values the values for which the variance is taken
00039 @param mean the average of values
00040 */
00041 double variance(std::vector<double> const &values, double mean);
00042
00043
00044 /*
00045 Calculates the average of a vector of doubles
00046 @param values the list of values for which the average is taken
00047 */
00048 double mean(std::vector<double> const &values);
00049
00050 /*
00051 Calculates the covariance of a set of points (needed for linear regression)
00052 https://en.wikipedia.org/wiki/Covariance
00053
00054 @param points the points for which the covariance is taken
00055 @param meanx the mean value of all x coordinates in points
00056 @param meany the mean value of all y coordinates in points
00057 */
00058 double covariance(std::vector<std::pair<double, double> const &points, double meanx, double meany);
00059
00060 /*
00061 Calculates the slope and y intercept of the line of best fit for the data
00062 @param points the points for the data
00063 */
00064 std::pair<double, double> calculate_linear_regression(std::vector<std::pair<double, double> const &points);
00065
00066 double estimate_path_length(const std::vector<point_t> &points);

```

## 6.31 moving\_average.h

```

00001 #pragma once
00002 #include <vector>
00003
00008 class Filter
00009 {
0010 public:
0011     virtual void add_entry(double n) = 0;
0012     virtual double get_value() const = 0;
0013 };
0014
0027 class MovingAverage : public Filter
0028 {
0029 public:
0030     /*
0031      * Create a moving average calculator with 0 as the default value
0032      *
0033      * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
0034      * reading
0035      */
0035     MovingAverage(int buffer_size);
0036     /*
0037      * Create a moving average calculator with a specified default value
0038      * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
0039      * reading
0040      * @param starting_value The value that the average will be before any data is added
0041      */
0041     MovingAverage(int buffer_size, double starting_value);
0042
0043     /*
0044      * Add a reading to the buffer
0045      * Before:
0046      * [ 1 1 2 2 3 3 ] => 2
0047      * ^
0048      * After:
0049      * [ 2 1 2 2 3 3 ] => 2.16
0050      * ^
0051      * @param n   the sample that will be added to the moving average.
0052      */
0053     void add_entry(double n);
0054
0055     double get_value() const;
0056
0057     int get_size() const;
0058
0059 private:
0060     int buffer_index;           // index of the next value to be overridden
0061     std::vector<double> buffer; // all current data readings we've taken
0062     double current_avg;        // the current value of the data
0063 };
0064
0085 class ExponentialMovingAverage : public Filter
0086 {
0087 public:
0088     /*
0089      * Create a moving average calculator with 0 as the default value
0090      *
0091      * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
0092      * reading
0093      */
0093     ExponentialMovingAverage(int buffer_size);
0094     /*
0095      * Create a moving average calculator with a specified default value
0096      * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
0097      * reading
0098      * @param starting_value The value that the average will be before any data is added
0099      */
0099     ExponentialMovingAverage(int buffer_size, double starting_value);
0100
0101     /*
0102      * Add a reading to the buffer
0103      * Before:
0104      * [ 1 1 2 2 3 3 ] => 2
0105      * ^
0106      * After:
0107      * [ 2 1 2 2 3 3 ] => 2.16
0108      * ^
0109      * @param n   the sample that will be added to the moving average.
0110      */
0111     void add_entry(double n);
0112
0117     double get_value();
0118
0123     int get_size();
0124
0125 private:

```

```

00126     int buffer_index;           // index of the next value to be overridden
00127     std::vector<double> buffer; // all current data readings we've taken
00128     double current_avg;       // the current value of the data
00129 };

```

## 6.32 pure\_pursuit.h

```

00001 #pragma once
00002
00003 #include <vector>
00004 #include "../core/include/utils/geometry.h"
00005 #include "../core/include/utils/vector2d.h"
00006 #include "vex.h"
00007
00008 using namespace vex;
00009
00010 namespace PurePursuit {
00014     class Path
00015     {
00016         public:
00022         Path(std::vector<point_t> points, double radius);
00023
00027         std::vector<point_t> get_points();
00028
00032         double get_radius();
00033
00037         bool is_valid();
00038
00039     private:
00040         std::vector<point_t> points;
00041         double radius;
00042         bool valid;
00043     };
00048     struct spline
00049     {
00050         double a, b, c, d, x_start, x_end;
00051
00052         double getY(double x) {
00053             return a * pow((x - x_start), 3) + b * pow((x - x_start), 2) + c * (x - x_start) + d;
00054         }
00055     };
00060     struct hermite_point
00061     {
00062         double x;
00063         double y;
00064         double dir;
00065         double mag;
00066
00067         point_t getPoint() const {
00068             return {x, y};
00069         }
00070
00071         Vector2D getTangent() const {
00072             return Vector2D(dir, mag);
00073         }
00074     };
00075
00080     extern std::vector<point_t> line_circle_intersections(point_t center, double r, point_t point1,
00081     point_t point2);
00084     extern point_t get_looking_ahead(const std::vector<point_t> &path, pose_t robot_loc, double radius);
00085
00089     extern std::vector<point_t> inject_path(const std::vector<point_t> &path, double spacing);
00090
00102     extern std::vector<point_t> smooth_path(const std::vector<point_t> &path, double weight_data, double
00103     weight_smooth, double tolerance);
00104
00104     extern std::vector<point_t> smooth_path_cubic(const std::vector<point_t> &path, double res);
00105
00114     extern std::vector<point_t> smooth_path_hermite(const std::vector<hermite_point> &path, double
00115     step);
00115
00126     extern double estimate_remaining_dist(const std::vector<point_t> &path, pose_t robot_pose, double
00127     radius);
00128 }

```

## 6.33 serializer.h

```

00001 #pragma once

```

```

00002 #include <algorithm>
00003 #include <map>
00004 #include <string>
00005 #include <vector>
00006 #include <stdio.h>
00007 #include <vex.h>
00008
00010 const char serialization_separator = '$';
00012 const std::size_t MAX_FILE_SIZE = 4096;
00013
00015 class Serializer
00016 {
00017 private:
00018     bool flush_always;
00019     std::string filename;
00020     std::map<std::string, int> ints;
00021     std::map<std::string, bool> bools;
00022     std::map<std::string, double> doubles;
00023     std::map<std::string, std::string> strings;
00024
00026     bool read_from_disk();
00027
00028 public:
00030     ~Serializer()
00031     {
00032         save_to_disk();
00033         printf("Saving %s\n", filename.c_str());
00034         fflush(stdout);
00035     }
00036
00040     explicit Serializer(const std::string &filename, bool flush_always = true) :
00041         flush_always(flush_always), filename(filename), ints({}), bools({}), doubles({}),
00042         strings({})
00043     {
00044         read_from_disk();
00045     }
00047     void save_to_disk() const;
00048
00050     void set_int(const std::string &name, int i);
00055     void set_bool(const std::string &name, bool b);
00060     void set_double(const std::string &name, double d);
00065     void set_string(const std::string &name, std::string str);
00070
00073     int int_or(const std::string &name, int otherwise);
00079     bool bool_or(const std::string &name, bool otherwise);
00085     double double_or(const std::string &name, double otherwise);
00091     std::string string_or(const std::string &name, std::string otherwise);
00097 };

```

## 6.34 vector2d.h

```

00001 #pragma once
00002
00003
00004 #include <cmath>
00005 #include "../core/include/utils/geometry.h"
00006
00007 #ifndef PI
00008 #define PI 3.141592654
00009 #endif
00015 class Vector2D
00016 {
00017 public:
00024     Vector2D(double dir, double mag);
00025
00031     Vector2D(point_t p);
00032
00040     double get_dir() const;
00041
00045     double get_mag() const;
00046
00050     double get_x() const;
00051
00055     double get_y() const;

```

```
00056
00061     Vector2D normalize();
00062
00067     point_t point();
00068
00074     Vector2D operator*(const double &x);
00081     Vector2D operator+(const Vector2D &other);
00088     Vector2D operator-(const Vector2D &other);
00089
00090 private:
00091     double dir, mag;
00093
00094 };
00095
00101 double deg2rad(double deg);
00102
00109 double rad2deg(double r);
```



# Index

accel  
    OdometryBase, 97  
add  
    CommandController, 29  
    GenericAuto, 64  
add\_async  
    GenericAuto, 64  
add\_cancel\_func  
    CommandController, 31  
add\_delay  
    CommandController, 31  
    GenericAuto, 64  
add\_entry  
    ExponentialMovingAverage, 44  
    Filter, 51  
    MovingAverage, 88  
add\_samples  
    GraphDrawer, 66  
AndCondition, 11  
    test, 11  
ang\_accel\_deg  
    OdometryBase, 97  
ang\_speed\_deg  
    OdometryBase, 97  
Async, 12  
    run, 13  
auto\_drive  
    MecanumDrive, 79  
auto\_turn  
    MecanumDrive, 80  
AutoChooser, 13  
    AutoChooser, 14  
    choice, 15  
    draw, 14  
    get\_choice, 14  
    list, 15  
    update, 15  
AutoChooser::entry\_t, 42  
    name, 43  
AutoCommand, 16  
    on\_timeout, 17  
    run, 17  
    timeout\_seconds, 17  
background\_task  
    OdometryBase, 94  
BangBang, 18  
    get, 18  
    init, 18  
    is\_on\_target, 19  
set\_limits, 19  
update, 19  
BasicSolenoidSet, 20  
    BasicSolenoidSet, 20  
    run, 21  
BasicSpinCommand, 21  
    BasicSpinCommand, 22  
    run, 23  
BasicStopCommand, 23  
    BasicStopCommand, 24  
    run, 24  
bool\_or  
    Serializer, 129  
Branch, 25  
    on\_timeout, 26  
    run, 26  
ButtonWidget  
    screen::ButtonWidget, 27  
calculate  
    FeedForward, 48  
    TrapezoidProfile, 154  
calculate\_time\_based  
    TrapezoidProfile, 154  
choice  
    AutoChooser, 15  
CommandController, 28  
    add, 29  
    add\_cancel\_func, 31  
    add\_delay, 31  
    CommandController, 28  
    last\_command\_timed\_out, 31  
    run, 31  
Condition, 32  
control\_continuous  
    Lift< T >, 71  
control\_manual  
    Lift< T >, 71  
control\_setpoints  
    Lift< T >, 71  
Core, 1  
current\_pos  
    OdometryBase, 97  
CustomEncoder, 32  
    CustomEncoder, 33  
    position, 33  
    rotation, 33  
    setPosition, 34  
    setRotation, 34  
    velocity, 34

DelayCommand, 35  
     DelayCommand, 36  
     run, 36  
 dist  
     point\_t, 120  
 double\_or  
     Serializer, 129  
 draw  
     AutoChooser, 14  
     FlywheelPage, 55  
     GraphDrawer, 66  
     screen::FunctionPage, 63  
     screen::OdometryPage, 99  
     screen::Page, 106  
     screen::PIDPage, 119  
     screen::StatsPage, 137  
 drive  
     MecanumDrive, 80  
 drive\_arena  
     TankDrive, 141  
 drive\_forward  
     TankDrive, 142  
 drive\_raw  
     MecanumDrive, 81  
 drive\_tank  
     TankDrive, 143  
 drive\_to\_point  
     TankDrive, 143, 144  
 DriveForwardCommand, 36  
     DriveForwardCommand, 37  
     on\_timeout, 38  
     run, 38  
 DriveStopCommand, 38  
     DriveStopCommand, 39  
     on\_timeout, 39  
     run, 40  
 DriveToPointCommand, 40  
     DriveToPointCommand, 41  
     run, 42  
 end\_async  
     OdometryBase, 94  
 ERROR\_TYPE  
     PID, 111  
 ExponentialMovingAverage, 43  
     add\_entry, 44  
     ExponentialMovingAverage, 43, 44  
     get\_size, 44  
     get\_value, 44  
 Feedback, 45  
     get, 46  
     init, 46  
     is\_on\_target, 46  
     set\_limits, 46  
     update, 47  
 FeedForward, 47  
     calculate, 48  
     FeedForward, 48  
 FeedForward::ff\_config\_t, 49  
     kA, 49  
     kG, 49  
     kS, 50  
     kV, 50  
 Filter, 50  
     add\_entry, 51  
     get\_value, 51  
 Flywheel, 51  
     Flywheel, 52  
     get\_motors, 52  
     get\_target, 52  
     getRPM, 52  
     is\_on\_target, 53  
     Page, 53  
     spin\_manual, 53  
     spin\_rpm, 54  
     SpinRpmCmd, 54  
     spinRPMTask, 55  
     stop, 54  
     WaitUntilUpToSpeedCmd, 54  
 FlywheelPage, 55  
     draw, 55  
     update, 55  
 FlywheelStopCommand, 56  
     FlywheelStopCommand, 57  
     run, 57  
 FlywheelStopMotorsCommand, 57  
     FlywheelStopMotorsCommand, 58  
     run, 58  
 FlywheelStopNonTasksCommand, 59  
 FunctionCommand, 60  
     run, 61  
 FunctionCondition, 61  
     test, 62  
 FunctionPage  
     screen::FunctionPage, 62  
 GenericAuto, 63  
     add, 64  
     add\_async, 64  
     add\_delay, 64  
     run, 65  
 get  
     BangBang, 18  
     Feedback, 46  
     MotionController, 84  
     PID, 112  
     PIDFF, 116  
     TakeBackHalf, 138  
 get\_accel  
     OdometryBase, 94  
 get-angular\_accel\_deg  
     OdometryBase, 94  
 get-angular\_speed\_deg  
     OdometryBase, 95  
 get\_async  
     Lift< T >, 72  
 get\_choice

AutoChooser, 14  
get\_dir  
    Vector2D, 161  
get\_error  
    PID, 112  
get\_mag  
    Vector2D, 161  
get\_motion  
    MotionController, 84  
get\_motors  
    Flywheel, 52  
get\_movement\_time  
    TrapezoidProfile, 154  
get\_points  
    PurePursuit::Path, 109  
get\_position  
    OdometryBase, 95  
get\_radius  
    PurePursuit::Path, 109  
get\_sensor\_val  
    PID, 112  
get\_setpoint  
    Lift< T >, 72  
get\_size  
    ExponentialMovingAverage, 44  
    MovingAverage, 88  
get\_speed  
    OdometryBase, 95  
get\_target  
    Flywheel, 52  
    PID, 112  
get\_value  
    ExponentialMovingAverage, 44  
    Filter, 51  
    MovingAverage, 88  
get\_x  
    Vector2D, 161  
get\_y  
    Vector2D, 161  
getRPM  
    Flywheel, 52  
GraphDrawer, 65  
    add\_samples, 66  
    draw, 66  
    GraphDrawer, 65  
  
handle  
    OdometryBase, 98  
hold  
    Lift< T >, 72  
home  
    Lift< T >, 72  
  
IfTimePassed, 67  
    test, 68  
include/robot\_specs.h, 167  
include/subsystems/custom\_encoder.h, 167  
include/subsystems/flywheel.h, 168  
include/subsystems/lift.h, 168  
  
    include/subsystems/mecanum\_drive.h, 171  
    include/subsystems/odometry/odometry\_3wheel.h, 172  
    include/subsystems/odometry/odometry\_base.h, 172  
    include/subsystems/odometry/odometry\_tank.h, 173  
    include/subsystems/screen.h, 174  
    include/subsystems/tank\_drive.h, 176  
    include/utils/auto\_chooser.h, 177  
    include/utils/command\_structure/auto\_command.h, 177  
    include/utils/command\_structure/basic\_command.h,  
        179  
    include/utils/command\_structure/command\_controller.h,  
        180  
    include/utils/command\_structure/delay\_command.h,  
        181  
    include/utils/command\_structure/drive\_commands.h,  
        181  
    include/utils/command\_structure/flywheel\_commands.h,  
        183  
    include/utils/controls/bang\_bang.h, 184  
    include/utils/controls/feedback\_base.h, 184  
    include/utils/controls/feedforward.h, 184  
    include/utils/controls/motion\_controller.h, 185  
    include/utils/controls/pid.h, 185  
    include/utils/controls/pidff.h, 186  
    include/utils/controls/take\_back\_half.h, 187  
    include/utils/controls/trapezoid\_profile.h, 187  
    include/utils/generic\_auto.h, 188  
    include/utils/geometry.h, 189  
    include/utils/graph\_drawer.h, 190  
    include/utils/logger.h, 190  
    include/utils/math\_util.h, 191  
    include/utils/moving\_average.h, 192  
    include/utils/pure\_pursuit.h, 193  
    include/utils/serializer.h, 193  
    include/utils/vector2d.h, 194  
init  
    BangBang, 18  
    Feedback, 46  
    MotionController, 84  
    PID, 112  
    PIDFF, 116  
    TakeBackHalf, 138  
InOrder, 68  
    on\_timeout, 69  
    run, 69  
int\_or  
    Serializer, 129  
is\_on\_target  
    BangBang, 19  
    Feedback, 46  
    Flywheel, 53  
    MotionController, 85  
    PID, 113  
    PIDFF, 116  
    TakeBackHalf, 139  
is\_valid  
    PurePursuit::Path, 109  
kA

FeedForward::ff\_config\_t, 49  
 kG FeedForward::ff\_config\_t, 49  
 kS FeedForward::ff\_config\_t, 50  
 KV FeedForward::ff\_config\_t, 50  
 last\_command\_timed\_out CommandController, 31  
 Lift Lift< T >, 70  
 Lift< T >, 70  
     control\_continuous, 71  
     control\_manual, 71  
     control\_setpoints, 71  
     get\_async, 72  
     get\_setpoint, 72  
     hold, 72  
     home, 72  
     Lift, 70  
     set\_async, 72  
     set\_position, 73  
     set\_sensor\_function, 73  
     set\_sensor\_reset, 73  
     set\_setpoint, 73  
 Lift< T >::lift\_cfg\_t, 74  
 list AutoChooser, 15  
 Log Logger, 76  
 Logf Logger, 76  
 Logger, 74  
     Log, 76  
     Logf, 76  
     Logger, 75  
     LogIn, 77  
 LogIn  
     Logger, 77  
 Mat2, 78  
 MecanumDrive, 78  
     auto\_drive, 79  
     auto\_turn, 80  
     drive, 80  
     drive\_raw, 81  
     MecanumDrive, 79  
 MecanumDrive::mecanumdrive\_config\_t, 82  
 modify\_inputs  
     TankDrive, 145  
 motion\_t, 82  
 MotionController, 83  
     get, 84  
     get\_motion, 84  
     init, 84  
     is\_on\_target, 85  
     MotionController, 84  
     set\_limits, 85  
     tune\_feedforward, 85  
     update, 86  
 MotionController::m\_profile\_cfg\_t, 77  
 MovingAverage, 86  
     add\_entry, 88  
     get\_size, 88  
     get\_value, 88  
     MovingAverage, 87  
 mut OdometryBase, 98  
 name AutoChooser::entry\_t, 43  
 normalize Vector2D, 161  
 Odometry3Wheel, 89  
     Odometry3Wheel, 90  
     tune, 91  
     update, 91  
 Odometry3Wheel::odometry3wheel\_cfg\_t, 92  
     off\_axis\_center\_dist, 92  
     wheel\_diam, 92  
     wheelbase\_dist, 92  
 OdometryBase, 92  
     accel, 97  
     ang\_accel\_deg, 97  
     ang\_speed\_deg, 97  
     background\_task, 94  
     current\_pos, 97  
     end\_async, 94  
     get\_accel, 94  
     get\_angular\_accel\_deg, 94  
     get\_angular\_speed\_deg, 95  
     get\_position, 95  
     get\_speed, 95  
     handle, 98  
     mut, 98  
     OdometryBase, 94  
     pos\_diff, 95  
     rot\_diff, 96  
     set\_position, 96  
     smallest\_angle, 96  
     speed, 98  
     update, 97  
     zero\_pos, 98  
 OdometryPage  
     screen::OdometryPage, 99  
 OdometryTank, 100  
     OdometryTank, 101, 102  
     set\_position, 103  
     update, 103  
 OdomSetPosition, 103  
     OdomSetPosition, 104  
     run, 105  
 off\_axis\_center\_dist  
     Odometry3Wheel::odometry3wheel\_cfg\_t, 92  
 on\_timeout AutoCommand, 17

Branch, 26  
DriveForwardCommand, 38  
DriveStopCommand, 39  
InOrder, 69  
Parallel, 108  
PurePursuitCommand, 123  
RepeatUntil, 126  
TurnDegreesCommand, 157  
TurnToHeadingCommand, 159  
operator+  
    point\_t, 120  
    Vector2D, 162  
operator-  
    point\_t, 121  
    Vector2D, 162  
operator\*  
    Vector2D, 162  
OrCondition, 105  
    test, 105  
  
Page  
    Flywheel, 53  
Parallel, 107  
    on\_timeout, 108  
    run, 108  
parallel\_runner\_info, 108  
Path  
    PurePursuit::Path, 109  
PID, 110  
    ERROR\_TYPE, 111  
    get, 112  
    get\_error, 112  
    get\_sensor\_val, 112  
    get\_target, 112  
    init, 112  
    is\_on\_target, 113  
    PID, 111  
    reset, 113  
    set\_limits, 113  
    set\_target, 113  
    update, 114  
PID::pid\_config\_t, 114  
PIDFF, 115  
    get, 116  
    init, 116  
    is\_on\_target, 116  
    set\_limits, 116  
    set\_target, 117  
    update, 117  
PIDPage  
    screen::PIDPage, 118  
point  
    Vector2D, 163  
point\_t, 119  
    dist, 120  
    operator+, 120  
    operator-, 121  
pos\_diff  
    OdometryBase, 95  
pose\_t, 121  
position  
    CustomEncoder, 33  
pure\_pursuit  
    TankDrive, 146  
PurePursuit::hermite\_point, 67  
PurePursuit::Path, 109  
    get\_points, 109  
    get\_radius, 109  
    is\_valid, 109  
    Path, 109  
PurePursuit::spline, 136  
PurePursuitCommand, 122  
    on\_timeout, 123  
    PurePursuitCommand, 123  
    run, 123  
  
Rect, 123  
RepeatUntil, 124  
    on\_timeout, 126  
    RepeatUntil, 125, 126  
    run, 126  
reset  
    PID, 113  
reset\_auto  
    TankDrive, 147  
robot\_specs\_t, 127  
rot\_diff  
    OdometryBase, 96  
rotation  
    CustomEncoder, 33  
run  
    Async, 13  
    AutoCommand, 17  
    BasicSolenoidSet, 21  
    BasicSpinCommand, 23  
    BasicStopCommand, 24  
    Branch, 26  
    CommandController, 31  
    DelayCommand, 36  
    DriveForwardCommand, 38  
    DriveStopCommand, 40  
    DriveToPointCommand, 42  
    FlywheelStopCommand, 57  
    FlywheelStopMotorsCommand, 58  
    FunctionCommand, 61  
    GenericAuto, 65  
    InOrder, 69  
    OdomSetPosition, 105  
    Parallel, 108  
    PurePursuitCommand, 123  
    RepeatUntil, 126  
    SpinRPMCommand, 135  
    TurnDegreesCommand, 157  
    TurnToHeadingCommand, 159  
    WaitUntilCondition, 164  
    WaitUntilUpToSpeedCommand, 166  
save\_to\_disk

Serializer, 131  
 screen::ButtonWidget, 26  
   ButtonWidget, 27  
   update, 27  
 screen::FunctionPage, 62  
   draw, 63  
   FunctionPage, 62  
   update, 63  
 screen::OdometryPage, 98  
   draw, 99  
   OdometryPage, 99  
   update, 99  
 screen::Page, 106  
   draw, 106  
   update, 106  
 screen::PIDPage, 118  
   draw, 119  
   PIDPage, 118  
   update, 119  
 screen::ScreenData, 127  
 screen::SliderWidget, 133  
   SliderWidget, 133  
   update, 133  
 screen::StatsPage, 136  
   draw, 137  
   StatsPage, 137  
   update, 137  
 Serializer, 128  
   bool\_or, 129  
   double\_or, 129  
   int\_or, 129  
   save\_to\_disk, 131  
 Serializer, 128  
   set\_bool, 131  
   set\_double, 131  
   set\_int, 131  
   set\_string, 132  
   string\_or, 132  
 set\_accel  
   TrapezoidProfile, 155  
 set\_async  
   Lift< T >, 72  
 set\_bool  
   Serializer, 131  
 set\_double  
   Serializer, 131  
 set\_endpts  
   TrapezoidProfile, 155  
 set\_int  
   Serializer, 131  
 set\_limits  
   BangBang, 19  
   Feedback, 46  
   MotionController, 85  
   PID, 113  
   PIDFF, 116  
   TakeBackHalf, 139  
 set\_max\_v  
   TrapezoidProfile, 155  
 set\_position  
   Lift< T >, 73  
   OdometryBase, 96  
   OdometryTank, 103  
 set\_sensor\_function  
   Lift< T >, 73  
 set\_sensor\_reset  
   Lift< T >, 73  
 set\_setpoint  
   Lift< T >, 73  
 set\_string  
   Serializer, 132  
 set\_target  
   PID, 113  
   PIDFF, 117  
 set\_vel\_endpts  
   TrapezoidProfile, 155  
 setPosition  
   CustomEncoder, 34  
 setRotation  
   CustomEncoder, 34  
 SliderWidget  
   screen::SliderWidget, 133  
 smallest\_angle  
   OdometryBase, 96  
 speed  
   OdometryBase, 98  
 spin\_manual  
   Flywheel, 53  
 spin\_rpm  
   Flywheel, 54  
 SpinRpmCmd  
   Flywheel, 54  
 SpinRPMCommand, 134  
   run, 135  
   SpinRPMCommand, 135  
 spinRPTMTask  
   Flywheel, 55  
 StatsPage  
   screen::StatsPage, 137  
 stop  
   Flywheel, 54  
   TankDrive, 147  
 string\_or  
   Serializer, 132  
 TakeBackHalf, 138  
   get, 138  
   init, 138  
   is\_on\_target, 139  
   set\_limits, 139  
   update, 139  
 TankDrive, 140  
   drive\_arcade, 141  
   drive\_forward, 142  
   drive\_tank, 143  
   drive\_to\_point, 143, 144  
   modify\_inputs, 145

pure\_pursuit, 146  
reset\_auto, 147  
stop, 147  
TankDrive, 141  
turn\_degrees, 147, 149  
turn\_to\_heading, 150  
test  
    AndCondition, 11  
    FunctionCondition, 62  
    IfTimePassed, 68  
    OrCondition, 105  
    TimesTestedCondition, 151  
timeout\_seconds  
    AutoCommand, 17  
TimesTestedCondition, 151  
    test, 151  
trapezoid\_profile\_segment\_t, 152  
TrapezoidProfile, 152  
    calculate, 154  
    calculate\_time\_based, 154  
    get\_movement\_time, 154  
    set\_accel, 155  
    set\_endpts, 155  
    set\_max\_v, 155  
    set\_vel\_endpts, 155  
    TrapezoidProfile, 153  
tune  
    Odometry3Wheel, 91  
tune\_feedforward  
    MotionController, 85  
turn\_degrees  
    TankDrive, 147, 149  
turn\_to\_heading  
    TankDrive, 150  
TurnDegreesCommand, 156  
    on\_timeout, 157  
    run, 157  
    TurnDegreesCommand, 157  
TurnToHeadingCommand, 158  
    on\_timeout, 159  
    run, 159  
    TurnToHeadingCommand, 158  
  
update  
    AutoChooser, 15  
    BangBang, 19  
    Feedback, 47  
    FlywheelPage, 55  
    MotionController, 86  
    Odometry3Wheel, 91  
    OdometryBase, 97  
    OdometryTank, 103  
    PID, 114  
    PIDFF, 117  
    screen::ButtonWidget, 27  
    screen::FunctionPage, 63  
    screen::OdometryPage, 99  
    screen::Page, 106  
    screen::PIDPage, 119  
    screen::SliderWidget, 133  
    screen::StatsPage, 137  
    TakeBackHalf, 139  
    Vector2D, 159  
        get\_dir, 161  
        get\_mag, 161  
        get\_x, 161  
        get\_y, 161  
        normalize, 161  
        operator+, 162  
        operator-, 162  
        operator\*, 162  
        point, 163  
        Vector2D, 160  
velocity  
    CustomEncoder, 34  
WaitUntilCondition, 163  
    run, 164  
WaitUntilUpToSpeedCmd  
    Flywheel, 54  
WaitUntilUpToSpeedCommand, 164  
    run, 166  
    WaitUntilUpToSpeedCommand, 165  
wheel\_diam  
    Odometry3Wheel::odometry3wheel\_cfg\_t, 92  
wheelbase\_dist  
    Odometry3Wheel::odometry3wheel\_cfg\_t, 92  
zero\_pos  
    OdometryBase, 98