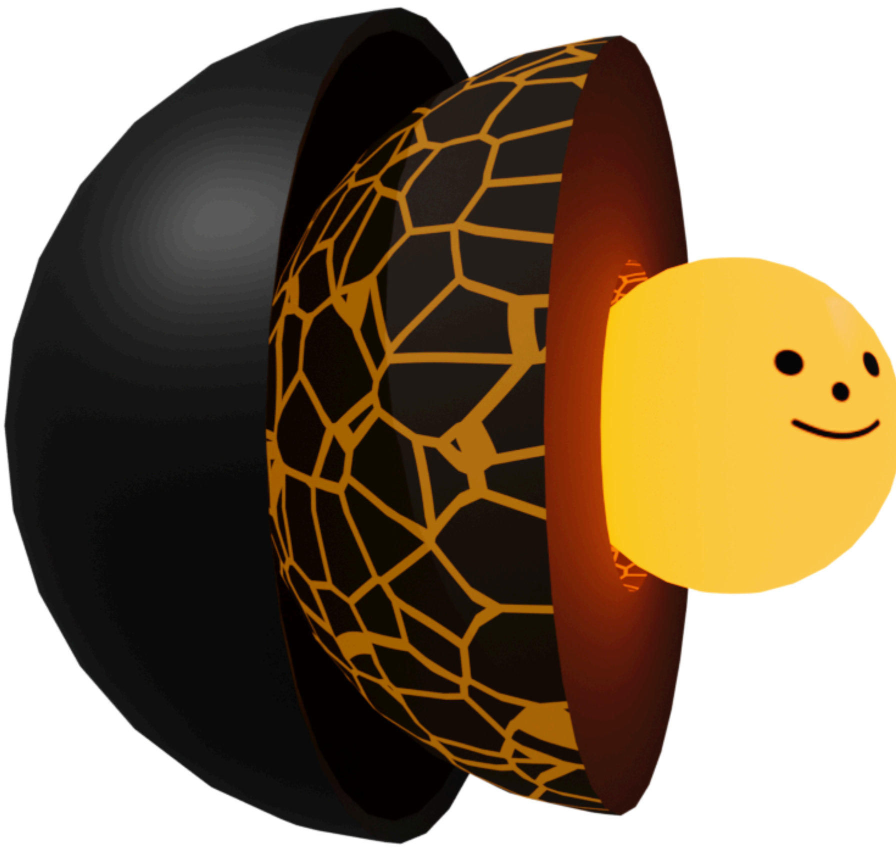
RIT VEXU Software Engineering Notebook

2023-2024



Overview

Core API

All of the robot code we use is built on top of our own custom library, called the Core API, which itself is built on top of the official VEX V5 library. This API contains template code for common subsystems such as drivetrains, lifts, flywheels, and odometry, and common utilities such as vector math and command-based autonomous functions. This code remains persistent between years and is constantly updated and improved. The library can be found at ***github.com/RIT-VEX-U/Core***

The RIT codebase is abstracted in a way that allows for simple use during a hectic build season, and creates a solid foundation for future expansion. Subsystems are divided into layers following an object-oriented approach to software development.

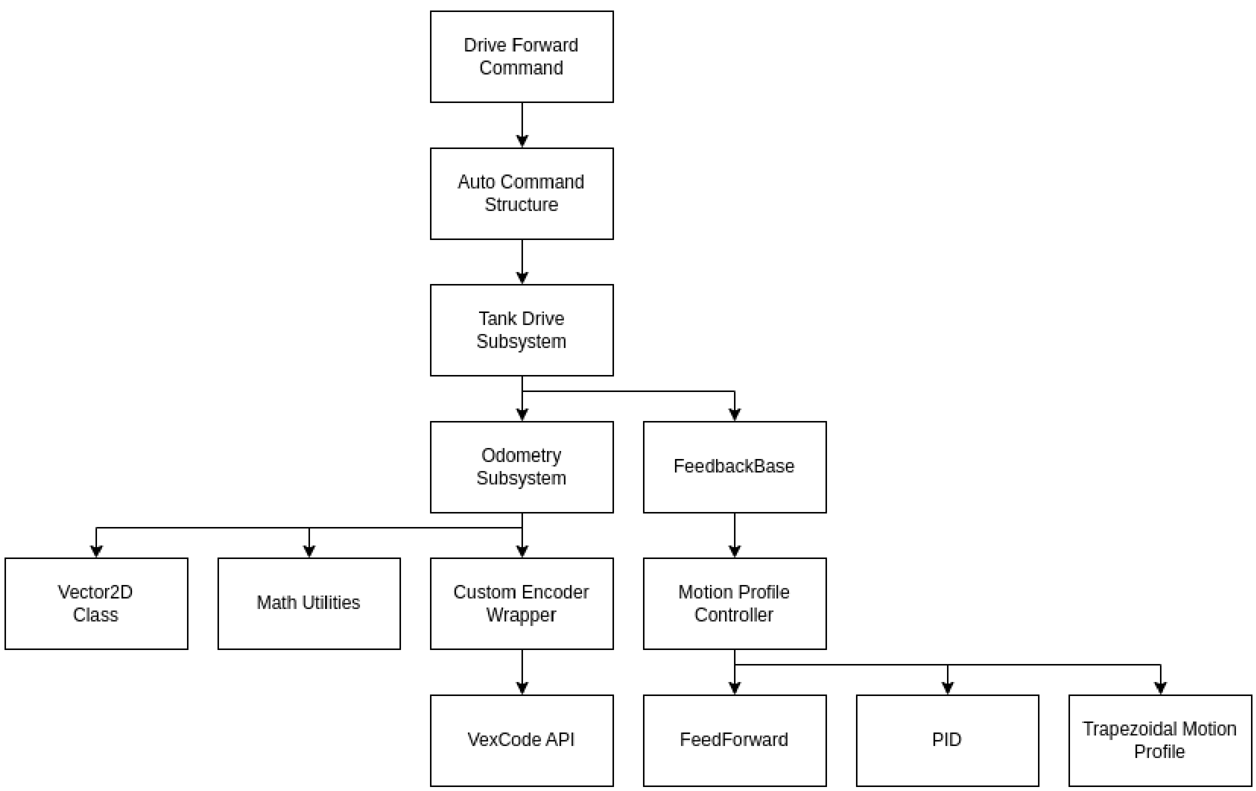


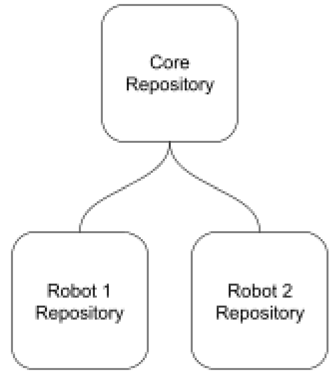
Figure X - Example of Object-Oriented Programming

Open Source Software

The RIT Core API is under the MIT open-source license, and is open for other teams to use and improve upon via pull requests. This system was modeled after the Okapi library from the Pros ecosystem, and offers similar functionality for the VexCode ecosystem. Teams that use this API are also encouraged to open source their software.

Project Structure

During the season, there are three repositories that are actively developed. Two repositories for the two competition robots, and one for the Core API. Development and code building occurs in the robot repos, and any changes to shared code (drivetrain, math utilities, major subsystems) are merged with the Core repo. This method reduces redundant code and development time.



Git Subrepo

The Core API uses a unique type of version control called Git Subrepo (github.com/ingydotnet/git-subrepo). This allows users to simply clone the repository into an existing VexCode project to have instant access to all the tools. It also allows users to instantly receive updates by pulling from the main branch, and makes sharing code between two robot projects easier with git code merges.

Before choosing Subrepo, the team experimented with using Git Submodules to incorporate the Core API into projects. This however made Core development cumbersome and difficult for anyone unfamiliar with Git submodules specifically. Subrepo made inter-project merges more streamlined, and simplified development.

Github Project Board

In order for our software team to collaborate together with these projects, we use the Github Projects kanban-style project board. This allows us to create and assign tasks, link it to a repository and additionally notify the assigned programmer through a slackbot.

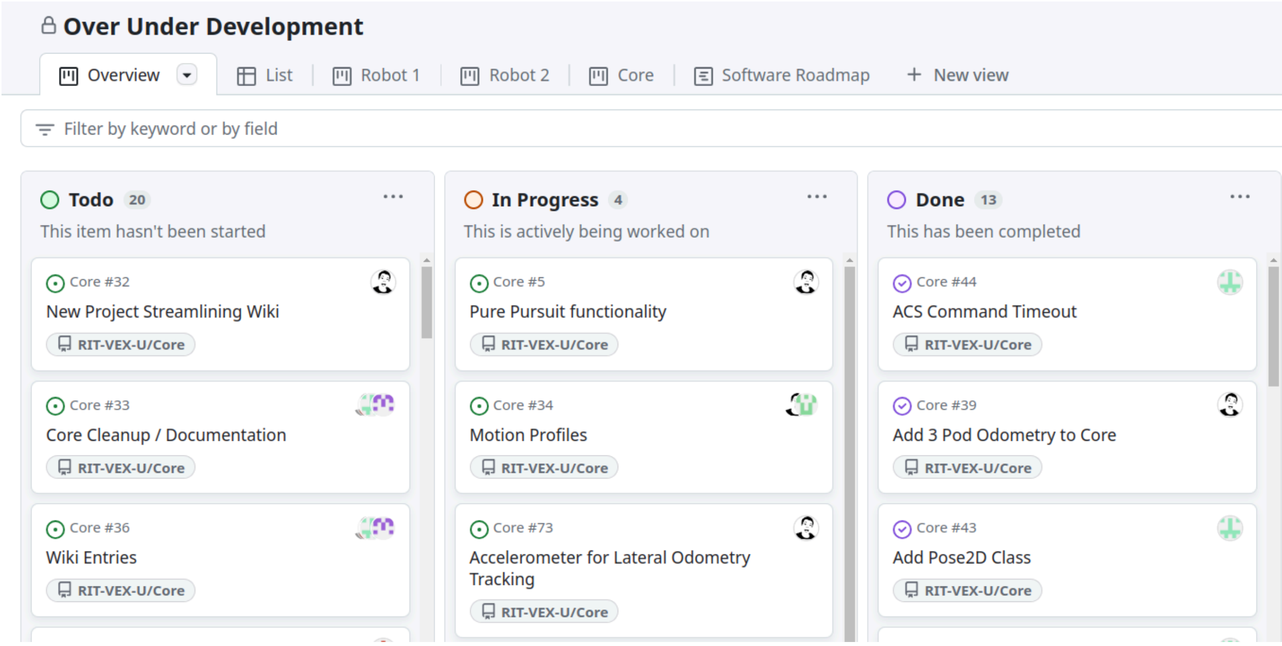
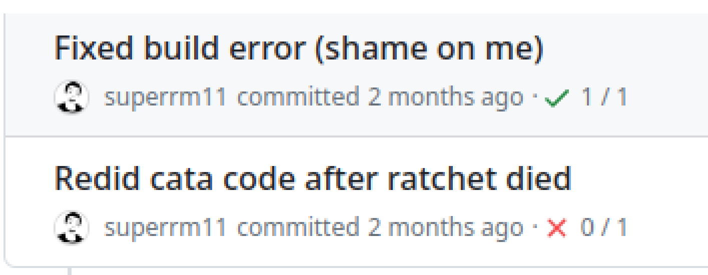


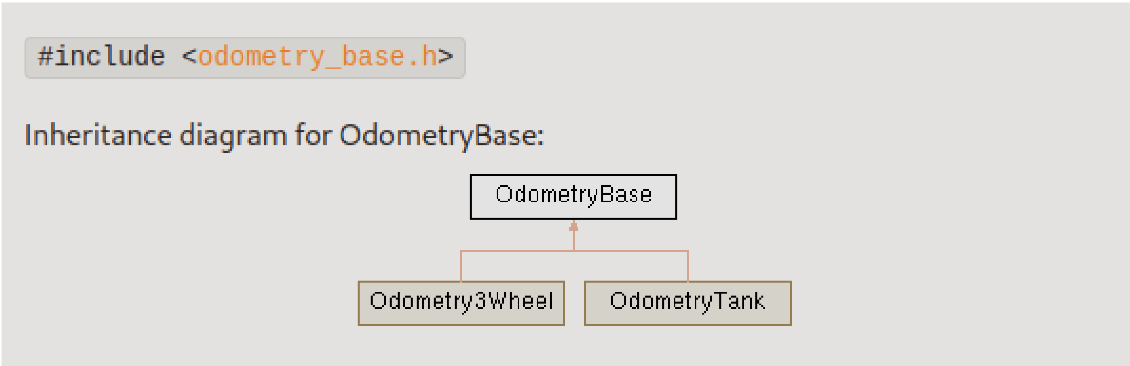
Figure X - Software Project Board

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.



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 or poor for performance and tests many other checks developed and validated by the wider C++ community.

Wiki

Whenever a new feature is added to Core, we create a Wiki page on the Core Github repository that provides documentation on what the function does, how to use it, and some examples of how it can be used. This documentation is easily accessible as it can be found online within the Core repository itself. This allows for new members to get acquainted with Core faster and easier than before. This allows us to speed up our training process and allow new members to start developing sooner rather than later. In addition it provides us and anyone using Core great documentation that not only goes into method signatures and descriptions, but also detailed explanations of what different methods, classes or functions do. An example of the Wiki is shown below.

Opcontrol

This class provides two ways of driving the robot with a controller: Tank drive and Arcade drive. Drivers can choose what they're most comfortable with.

Tank Drive - The left joystick controls the left-side motors, and the right joystick controls the right-side motors

Arcade Drive - Acts somewhat to how modern racing video games are controlled. The left joystick controls the forward / backward speed, and the right joystick controls turning left / right.

Both functions also have an optional parameter called `power`, and refers to how the joystick is scaled to the motors. The higher the power is, the more control you have over low-speed maneuvers. Because the scaling is non-linear, it may feel weird to those who aren't used to it.

Method Signatures

```
void drive_tank(double left, double right, int power=1);
void drive_arcade(double forward_back, double left_right, int power=1);
```

Usage Examples

```
drive_system.drive_tank(controller.Axis3.position() / 100.0, controller.Axis2.position() / 100.0);
drive_system.drive_arcade(controller.Axis3.position() / 100.0, controller.Axis1.position() / 100.0);
```

Core: Fundamentals

- Odometry
  - Tank (same old)
  - ~~GPS + Odometry~~
- Drivetrain
  - Tank Drive
    - What it do
      - DriveForward, TurnToHeading
      - DriveToPoint
    - Whats new
      - Pure pursuit
      - Brake mode

Odometry

Tank Odometry

// todo idk stuff here

GPS + Odometry

In order to fit an 8-motor drivetrain into the 15" size requirement, the robots could not fit undriven odometry wheels, leaving only the drive encoders to be used for position tracking. This isn't ideal, since sudden changes in acceleration and wheel slippage can easily cause the tracking to drift a substantial amount. To combat drifting, we looked to the GPS sensor for localization.

The GPS sensor uses a tag-based approach for localization, using a coded strip around the perimeter of the field to estimate position. Between pose estimates, the integrated IMU provides inertial information to estimate changes in position and heading for a constant flow of data, presumably using some sort of onboard Kalman filter. The pose (X, Y, Heading) data is sent back to the Brain over the smart port. In addition, the GPS provides a "quality" value, which is a percentage that increases when the camera can see a large amount of tape, and decreases when the camera is blocked and the IMU detects change in position over a period of time.

To properly characterize the GPS sensor, X/Y/Heading data points were gathered at different positions around the field, facing different headings. The following graphs show the data points on the 12' x 12' field grid. Distance error (in inches) to the actual measurements is shown by color.

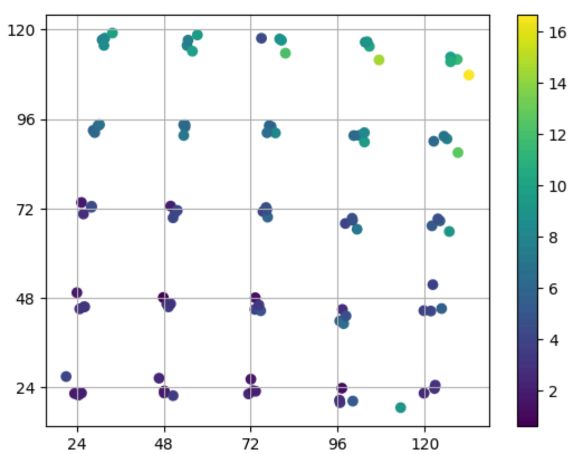


Figure X - Raw Data Points

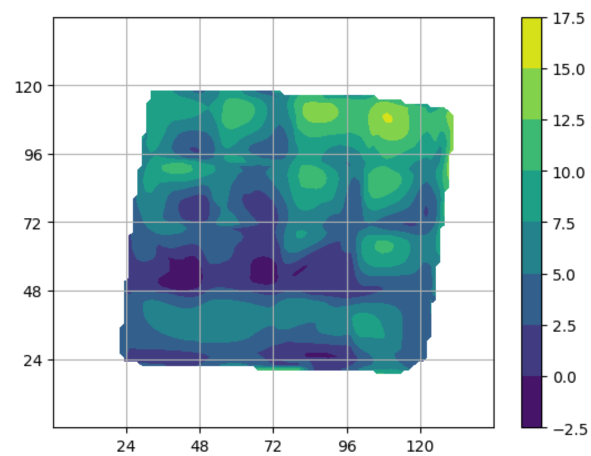


Figure X - Error Heat Map

There were some other errors with the GPS sensor, including issues localizing the robot when the sensor could not see enough of the coded tape. To take full advantage of the GPS sensor's localization capability, we'd need a way to perform sensor fusion alongside traditional ground-based odometry. To do this, a complementary filter was chosen - a filter that mixes two sets of data based on a proportional scalar *alpha* (α). the equation for a complementary filter is shown below:

$$out = \alpha * s_1 + (1 - \alpha) * s_2$$

where s_1 is *sensor 1*, s_2 is *sensor 2*, and alpha scales between the two.

To calculate alpha, first the X,Y position of the sensor and heading is taken into account. Since the robot will generally have a more accurate position when it's close to the wall and facing away from it, the following formula will report a score between 0 and 1 for the filter:

$$\alpha = (|\vec{c} - \vec{p}| \bullet |1|cos\theta) * \frac{\sqrt{(72-x)^2+(72-y)^2}}{101.8} * q$$

$$\alpha = \left(\frac{\vec{c} - \vec{p}}{||\vec{c}-\vec{p}||} \bullet \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \right) * \frac{||\vec{c} - \vec{p}||}{||\vec{c}||} * q$$

where $\vec{c}$ is the constant vector of a point in the center of the field (x=72in, y=72in), $\vec{p}$ is the robot's position as a vector (x, y), θ is the robot's heading, and x,y is again the robot's position (0 to 144 inches). The left side is the dot product of the normalized vector pointing from the robot to the center with the direction the robot is facing (gives 1 when the directions are aligned and -1 when opposite smoothly changing in between), and the right side is the sensor's distance to the center as a scalar percentage of the distance from corner to corner (between 0 and 1). Finally, q is the GPS's reported quality. We then remap this from the range [-1, 1] to [0, 1] when we do our mixing.

Drivetrain

A drivetrain class has two functions: To control the robot remotely, and autonomously. In the Core API, the TankDrive class allows the operator to control the robot using Tank controls (Left stick controls the left drive wheels, right controls the right), and Arcade controls (Left stick is forward / backwards, right stick is turning). This means drivers can tailor their controls to whichever feels more natural.

Tank Drivetrain

Brake Mode

A VEX driver has many things to keep track of during a match. From game element position, match load status, and partner robot condition there is a great deal going on. Defense is another layer on top of the mental load of playing the game. To ease this burden, we implemented a brake mode on our drive train. It is a multi-modal system that can either bring the robot to a stop or hold the robot in a specific location on the field. We use the motion profiles we developed for auto programming to decelerate the robot when requested and use our auto driving functions to hold the robot's position. We implement a smarter form of position holding than just motor braking as we can return to the exact location on the field. Additionally, we combine our deceleration control with position holding such that we do not immediately "lock the brakes" and skid away thus losing the position we attempt to hold and making driving incredibly difficult.

Autonomous Driving

For autonomous driving, the TankDrive class has multiple functions:

- drive_forward():
  - Drive X inches forward/back from the current position
  - Signature: drive_forward(double inches, directionType dir, double max_speed=1
- turn_degrees():
  - Drive X degrees CW/CCW from the current rotation
  - Signature: turn_degrees(double degrees, double max_speed=1)
- drive_to_point()
  - Drive to an absolute point on the field, using odometry
  - Signature: drive_to_point(double x, double y, vex::directionType dir, double max_speed=1);
- turn_to_heading()
  - Turn to an absolute heading relative to the field, using odometry
  - Signature: turn_to_heading(double heading_deg, double max_speed=1)

Generally, it is better to use drive_to_point and turn_to_heading to avoid compounding errors in position over relative movements. These functions implement the FeedbackBase class, so any control loop can be used to control it.

// Basic logic + async nature, feedback base class (use any feedback)
// DriveForward() - header declaration + examples
// TurnToHeading() - header decoration + examples

Drive To Point

The defining feature of a drive to point function is the ability for a robot to calculate a relative direction and distance between its own position and the target position, and navigate to it using tuned control loops. The steps taken for our implementation are listed below.

1 - Gather information

To drive towards a specific point, the robot must know the delta angle between the robot's heading and the target, and the distance to the target. To get this, we first grab the robot's current position and heading and create a positional difference vector between this and the new point.

```
pose_t current_pos = odometry->get_position();
pose_t end_pos = {.x = x, .y = y};

point_t pos_diff_pt =
{
 .x = x - current_pos.x,
 .y = y - current_pos.y
};

Vector2D point_vec(pos_diff_pt);
```

Using this information, grab the distance to the target (using a function in the Odometry subsystem). An issue with the pure distance between points is that it does not represent how far the robot has to travel to be considered "on target" in the control loop. In order to properly reach its target, the robot should report its "aligned distance", and ignore the lateral error, as per Figure X. This should only hold true when the robot is close to the target, or inside a given radius that is tuned by the user.

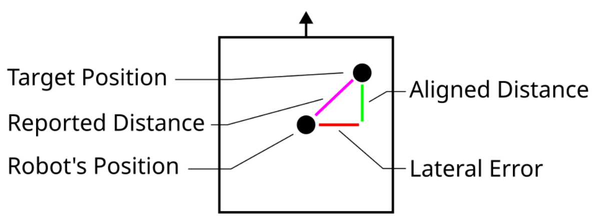


Figure X - Distance Modification

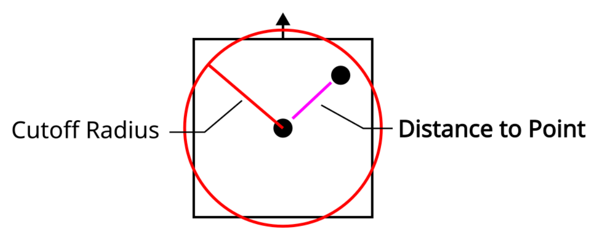


Figure X - Correction Cutoff Circle

```
double dist_left = OdometryBase::pos_diff(current_pos, end_pos);

if (fabs(dist_left) < config.drive_correction_cutoff)
{
 dist_left *= fabs(cos(angle * PI / 180.0));
}
```

The next data needed is the difference in angle between the robot's current heading and the vector between the robot's position and the target. This is calculated by using the arctangent of the difference vector, and subtracting it from the robot's current heading. The angle is then wrapped around 360 degrees.

```
double angle_to_point = atan2(y - current_pos.y, x - current_pos.x)
                        * 180.0 / PI;
double angle = fmod(current_pos.rot - angle_to_point, 360.0);
if (angle > 360)
 angle -= 360;
if (angle < 0)
 angle += 360;

double heading = rad2deg(point_vec.get_dir());
double delta_heading = 0;
if (dir == directionType::fwd)
 delta_heading = OdometryBase::smallest_angle(current_pos.rot, heading);
else
 delta_heading = OdometryBase::smallest_angle(current_pos.rot
                 - 180, heading);
```

The last piece of information needed is whether the robot should be moving forwards or backwards. Since the distance is calculated as $\sqrt{x^2 + y^2}$, the sign is lost when squaring. Re-implement the sign based on the angle and initial driving direction.

```
int sign = 1;
if (dir == directionType::fwd && angle > 90 && angle < 270)
 sign = -1;
else if (dir == directionType::rev && (angle < 90 || angle > 270))
 sign = -1;
```

2 - Setting Control Loops

In this section, the robot takes the above information and sets its feedback loops. Since the function takes in a FeedbackBase abstract class, any feedback can be used to drive the robot's correction and linear movements. The most common situation is a trapezoidal motion profile for linear distance with PD for heading correction.

// Logic & math - correction PID, distance cutoffs

// drive_to_point() - header declaration + examples

Pure Pursuit

// Logic & math - grab stuff from wiki, funny lookahead circle image
// pure_pursuit() - header declaration + examples

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 2π 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.

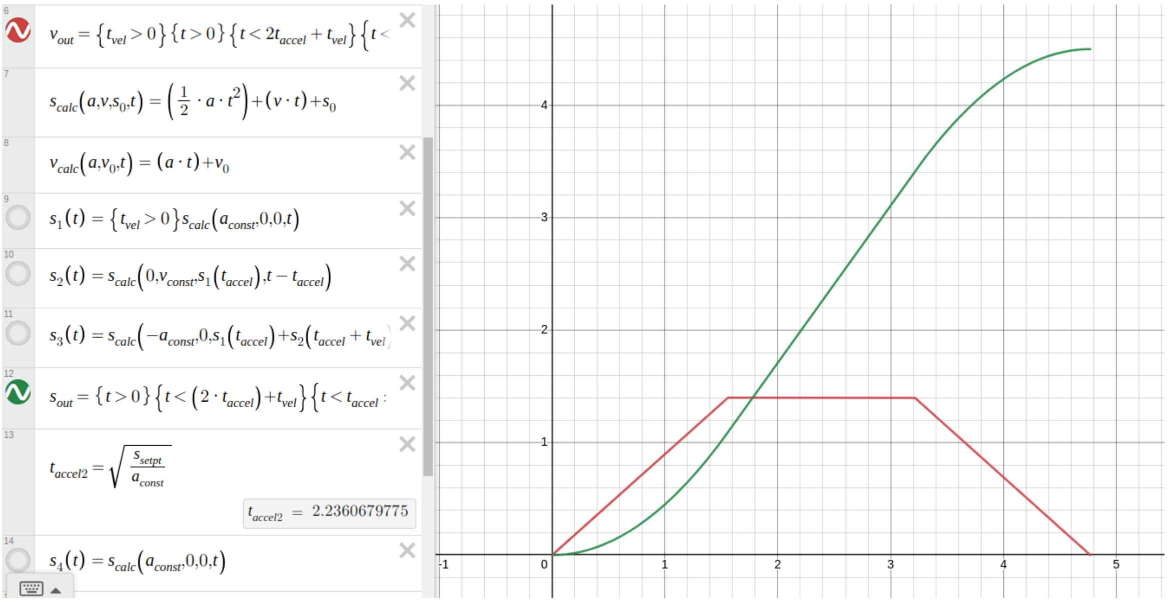
Generic Feedback

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 anonymous 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.

```cpp
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 often but for the most part 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 using a simple key-value interface. This keeps us from having to change a value, redeploy, repeat which cost us valuable time in the past.

Screen Subsystem

Principle

One of the most powerful elements of the V5 Brain is the fairly substantial touch screen. However, its simple drawing API limits its utility as one person's part of the code will draw over another since there is no larger abstraction controlling who draws when. We have many different subsystems on our robot to observe and debug and many parameters that can be tuned at run time and the screen provides a way to do this. We provide an API that provides a 'page' interface that can be inserted into a slideshow-like interface. Each 'page' provides two functions, an update and a draw. The update runs more frequently allowing touch input and data collection at a reasonably fast rate while the draw function runs less frequently to not cause too much overhead on the system. At startup, users provide the screen subsystem a list of pages and the screen subsystem handles orchestration and input in a background thread while other robot code runs unaffected.

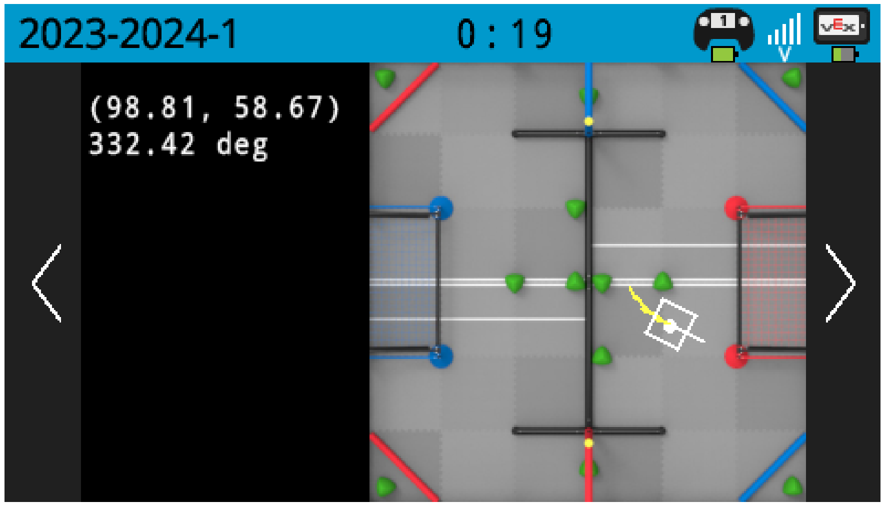
```
pages = {
    new AutoChooser({"Auto 1", "Auto 2", "Auto 3", "Auto 4"}),
    new screen::StatsPage(motor_names),
    new screen::OdometryPage(odom, 12, 12, true),
    cata_sys.Page(),
};

screen::start_screen(Brain.Screen, pages);
```

Configuration for the screen subsystem

Pages

Odometry Page

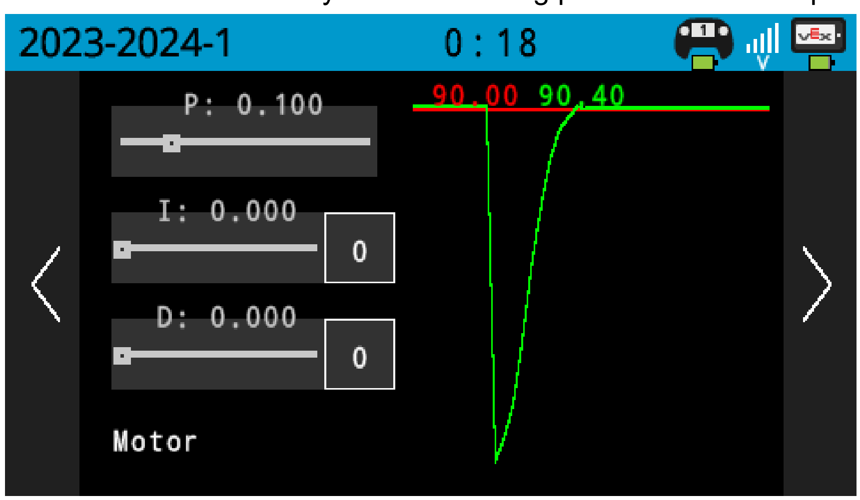
The odometry page has proved incredibly useful in writing and debugging auto and auto skills paths. It shows a picture of the robot on the field as well as a print out of the actual x,y coordinates and heading of the robot. Since we write our autos with respect to the coordinate system of the field, having a map to look at makes development much simpler.



A field display for the Over Under season

PID Tuner

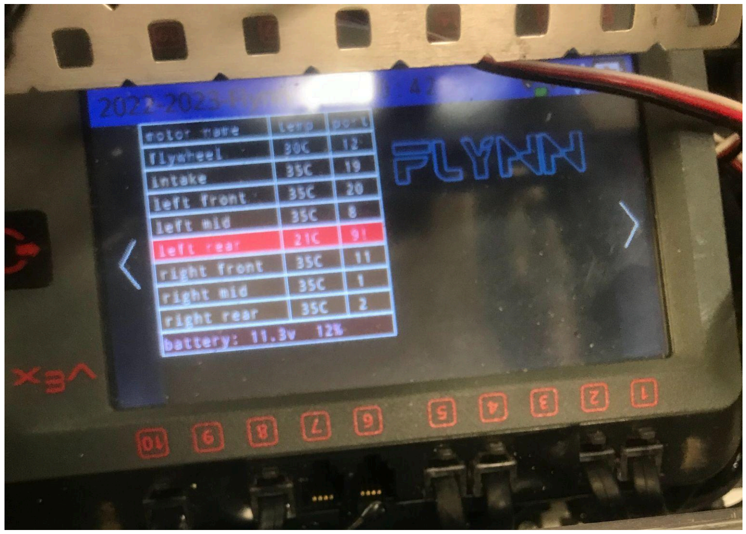
PID controllers are integral to many subsystems on our robot. Our drive code uses them for turning and forward motion, our catapult uses them for reloading, and subsystems across seasons require them for precise control. Tuning them, however, can be incredibly tedious. Changing one value, redeploying, and repeating over and over again is time consuming and unnecessary. Since we have a wonderful touchscreen, we simply added a series of sliders for PID parameters and we can now easily adjust a pid tuning in seconds rather than minutes saving a great deal of time on an already time consuming part of robot development.



Tuning a motor for reaching an angle

Motor Stats

One would think it an easy step to remember to plug motors in, and yet multiple times this season we have been bewildered and hindered by an unplugged motor. This page was written to continuously display that the motor had been unplugged and was not cancellable like the built-in VEX alert. This screen also displays what port to plug it into as well as a color coded temperature displaying when the robot needs to cool down. This tool proved extremely useful as we discovered an alarmingly high number of dead or nonfunctioning ports on the brain.



Motor Stats screen from our 2023 robot

Cata System Page

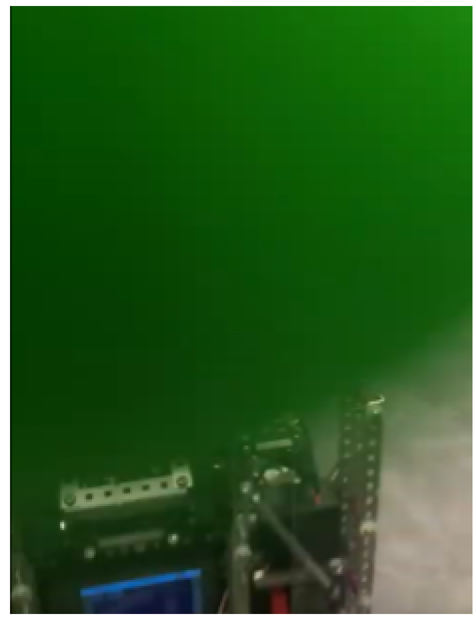
// get a picture of this when we get back

Catapult System

Vex's Over Under game requires the effective utilization of the fascinatingly shaped triball. After much deliberation our team decided on a catapult to launch the game element across the field and a reversible intake for picking up and scoring the triball. This system gives us a great deal of flexibility and power for strategy but does increase the complexity of the system. This complexity mostly stems from the orchestration of intaking with catapult reloading such that we do not jam our catapult and we never intake multiple game pieces leading to a disqualification.

We implemented a state machine that receives inputs from the controller, a distance sensor in the intake, a distance sensor in the catapult and a potentiometer for watching catapult position. The system runs the appropriate motors to either intake to hold the triball, reset the catapult, intake into the catapult, or shoot depending on what state it is in. Because we have so many sensors, we can determine when intaking would lead to disqualification and simply not honor the intaking message.

These messages are a simple enum that one passes to `CataSys::send_command()`. This was originally intended to make writing multi threaded code less error prone as there was one thread safe and simple way to interact with the subsystem rather than many disparate methods some of which are meant for internal usage of the class on the running thread and some accessors and setters meant to used from the user thread. Although it started for implementation ease, it naturally brought about a very simple interface for auto. Instead of sending a command on a button press, we simply send a command at a certain point in our auto path and the system reacts accordingly.



Though our system provides thread and disqualification safety, it does not provide programmer safety

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

Machine Learning

-

Rust

Over the course of the year, we have experimented with rewriting our Core API in Rust, a multi-paradigm programming language focused on performance and safety. Rust offers several potential advantages over C++:

Memory Safety

One of the primary benefits of Rust is its emphasis on memory safety without sacrificing performance. Rust's ownership model ensures that memory is managed correctly at compile time, reducing the risk of memory leaks and buffer overflows which are common issues in C++. This is especially crucial in robotics, where memory management errors can lead to system crashes or unpredictable behavior in real-time operations.

Concurrency

Rust's approach to concurrency is another major advantage. Concurrency errors, like race conditions, are hard to debug and can be catastrophic in robotics, leading to inconsistent states and erratic behavior. Rust's type system and ownership model prevent data races at compile time, making concurrent programming more reliable and easier to reason about.

Performance

In terms of performance, Rust is comparable to C++, which is essential in robotics where processing speed and response time are critical. Rust’s zero-cost abstractions mean that high-level constructs do not add overhead at runtime. This allows developers to write high-level code without compromising on performance, an important consideration in robotics where every millisecond can count.

Improved Code Maintenance and Readability

Rust also offers improved code maintainability and readability. Its modern syntax and language features make it easier to write clear and concise code. This reduces the cognitive load on developers, making it easier to develop and maintain complex robotic systems. The compiler's strictness also ensures that many potential bugs are caught early in the development cycle, reducing the time spent on debugging.

Growing Ecosystem and Community

The Rust ecosystem is rapidly growing, with a strong focus on safety and performance. There are increasing numbers of libraries and tools being developed for Rust, including those specifically for robotics. The Rust community is known for its dedication to improving code quality and security, which aligns well with the needs of robotics development.

Overall

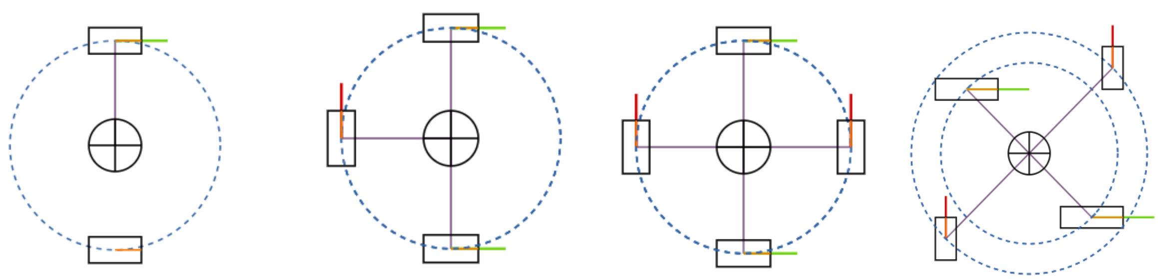
While the transition from C++ to Rust in a robotics context requires investment in terms of learning and codebase modification, the benefits in memory safety, concurrency handling, performance, and maintainability make it a compelling choice. The modern features of Rust, combined with its growing ecosystem and community support, position it well for developing robust, efficient, and safe robotic systems.

N-Pod Odometry

Motivation

Although we have been working on the GPS odometry system, wheel odometry is still vital. It provides great small scale, quickly updating positions as well as having near perfect continuous velocity which a GPS system can not achieve. We use odometry in two ways; either tank or differential odometry where there is one wheel on either side of the drivetrain alongside the drive wheels and 3 wheel odometry where we have 3 wheels at ninety degrees to each other. Tank odometry is limited as it can not track horizontal movement and we simply hope that we never move sideways, though it is easiest to implement in the robot so it is our most commonly used system. 3 wheel odometry solves the side-to-side problem but is much harder to implement in hardware owing to the extra wheel where other subsystems would need space.

In a plea for mercy from the hardware team, we agreed that we would take tracking wheels wherever and we could make do. Though we once again got stuck with a tank system, if our dream of more tracking wheels ever comes true we would need code to handle such a system. As well, since tank and 3 wheel odometry are special cases of an n-pod system, we could reduce code duplication.

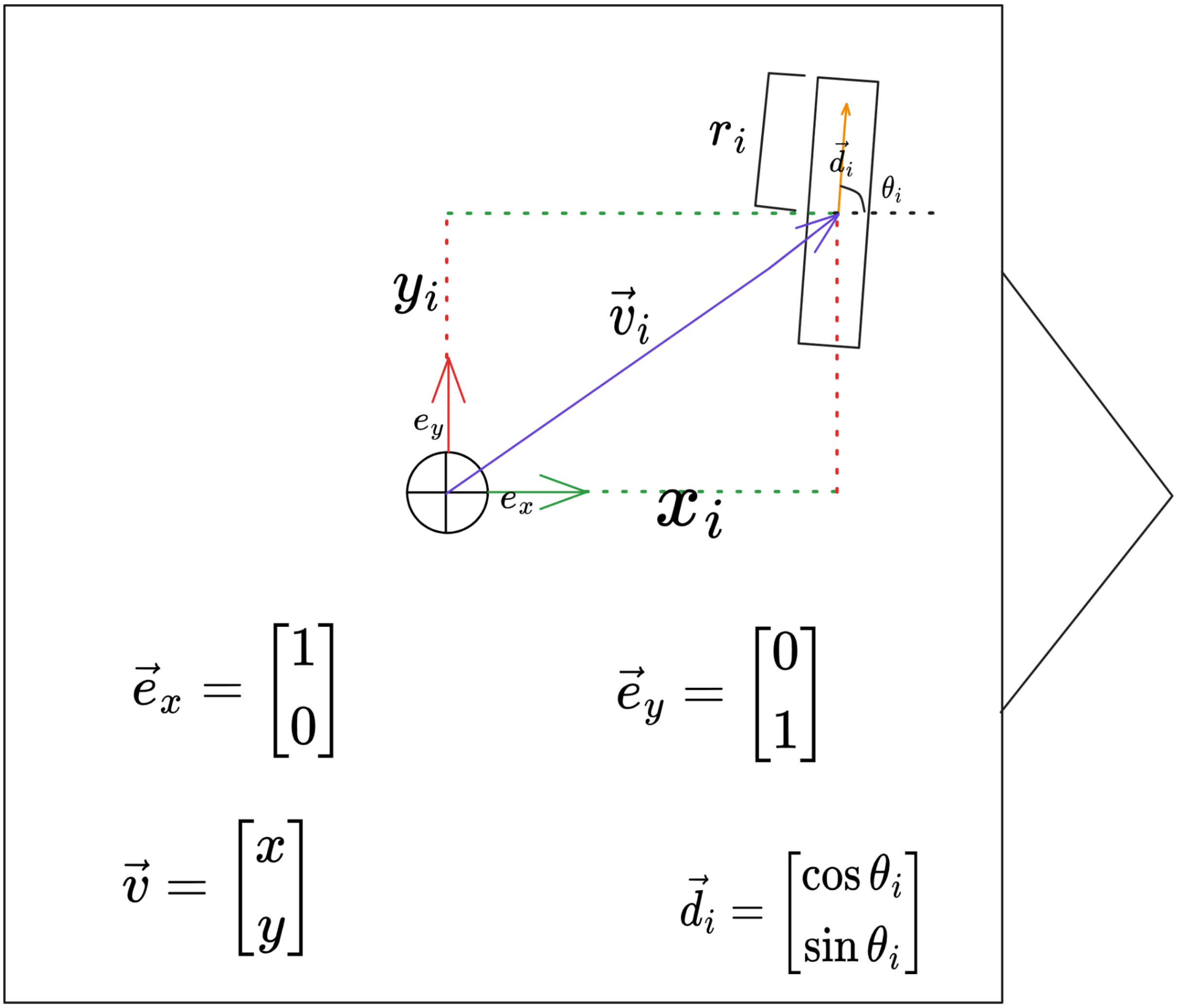


2 pod, 3 pod, and arbitrary pods such a system could handle.

Syntax

After much brainstorming and many mad scientists whiteboard drawings, we believe that we have the fundamentals of a system figured out. Unfortunately, other responsibilities to the team came up so we do not yet have a functioning implementation of the system.

Imagine a robot with *n* omni tracking wheels. We could read encoder values E_1, E_2, ..., E_3 from the system in radians from the initial position. As well, each encoder has a configuration $(x_1, y_1, \theta_1, r_1)$, …, $(x_n, y_n, \theta_n, r_1)$ describing its position relative to the center of rotation, an angle describing its orientation relative to the robot frame, and a radius of the wheel.



The configuration of a tracking wheel on the robot. (e_x, e_y) are the basis vectors of our coordinate system. X and Y axes of the robot coordinate frame

Now, if we pretend that these wheels are powered and we wish to translate and rotate the robot according to some controller input (x, y, θ) we can develop a formula for how much each wheel needs to rotate to move the robot in that direction with that rotation. Luckily, since the tracking wheels are omni wheels which roll freely in the axis against their “forward” direction, we do not need to worry about dragging a wheel so long as it is spinning the correct amount in its “forward” direction. For a desired (x, y, θ) (in the robots reference frame), for the i-th encoder we say $E_i = xF_i$. That is, for a movement in the x axis the rotations of the i-th encoder, are the desired x movement times some scalar factor for how far this specific wheel would rotate. Similarly, for a *y* only and θ only movement, $E_i = yF_{yi}$ and $E_i = \theta F_{\theta i}$ respectively.

Deriving Factors

F_x

F_x depends on the direction vector $\vec{d}$ of the omni wheel. If the omni wheel is facing along the x axis, F_x will be higher whereas if the omni wheel is directly perpendicular to the x axis, it will not spin when you move only in the x-direction. Since $\vec{e}_x$ and $\vec{d}$ are unit vectors, how closely they are related is given by $\vec{e}_x \cdot \vec{d} = \cos(angle\ between\ x\ axis\ and\ wheel)$

F_x also depends on the radius of the wheel r. One full rotation of the wheel moves a distance of $C = 2\pi r$. If we drive in the direction of the wheel *i* inches, the wheel will complete $\frac{i}{2\pi r}$ revolutions. If we measure the rotations in radians, the wheel will travel $\frac{i}{r}$ radians. That is, if the encoder wheel travels *E* radians, we will have traveled *Er* inches in that direction.

So, the distance traveled in the x direction of a wheel pointing in the direction $\vec{d}$, rotating *E* radians is $x = Er(\vec{e}_x \cdot \vec{d})$. This gives since F_x as how many inches per radian turned,

$$F_x = \frac{x}{E} = r(\vec{e}_x \cdot \vec{d})$$

F_y

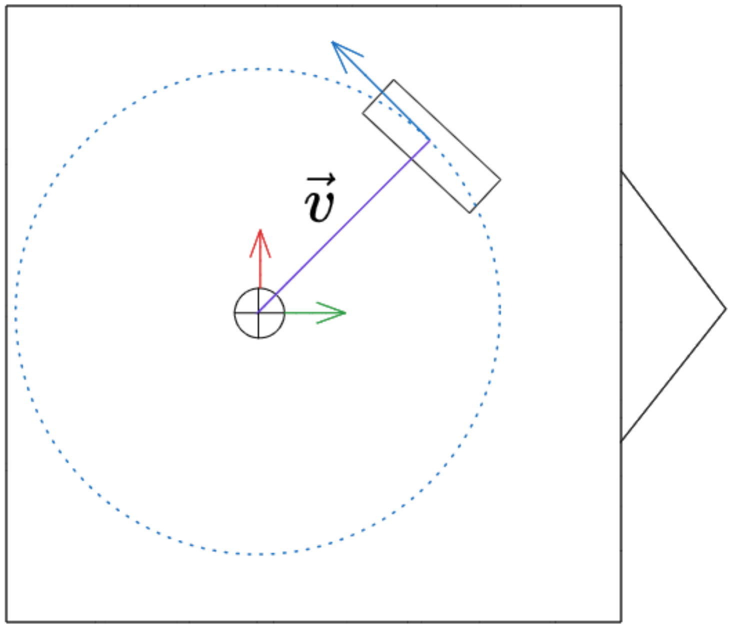
F_y is derived almost identically as F_x just instead of testing against $\vec{e}_x$ we test against $\vec{e}_y$. So,

$$F_y = \frac{y}{E} = r(\vec{e}_y \cdot \vec{d})$$

F_θ

F_θ is a little more complicated since it is determined by the position of the wheel $\vec{v}$ as well as the orientation of the wheel $\vec{v}$

Imagine that the robot turns an angle of θ_r measured in radians. A wheel that is perfectly perpendicular to the rotation will travel an arc with distance $S = ||\vec{v}||\,\theta_r$ by the arc length formula where the 'radius' of the arc is defined by the length of the vector $\vec{v}$.

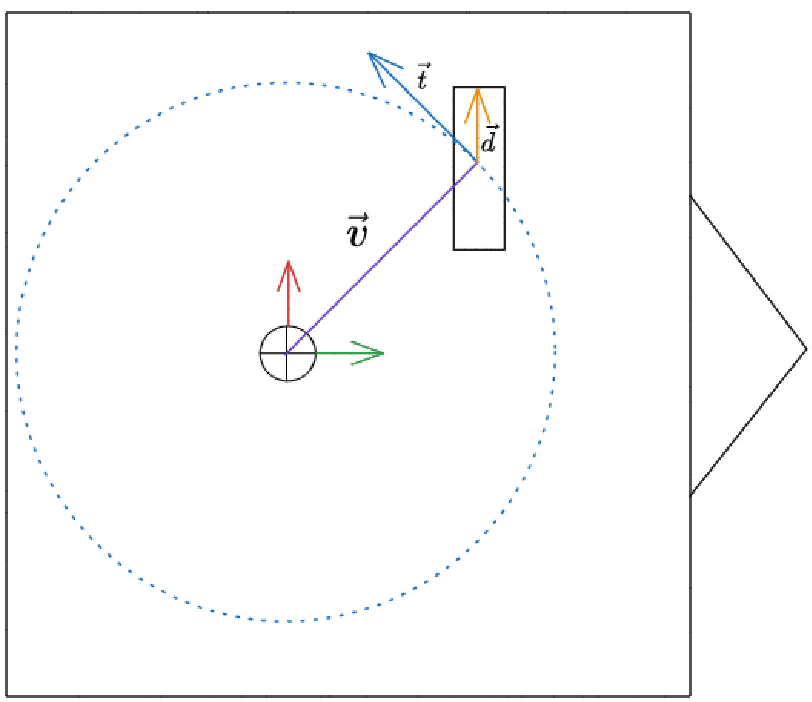


A conceptual perfectly perpendicular wheel

So, if we have a wheel that is always tangent to the rotation, it will travel

$E_t r = S = ||\vec{v}||\theta_r$

Since our wheel isn't guaranteed to be perfectly tangent to the arc, we have to use our dot product trick to get the component of its motion that is tangent to the turning circle. That is, instead of comparing to $\vec{e}_x$ or $\vec{e}_y$ we compare to the normalized vector $\vec{t}$ tangent to the turning circle.



$$E_t r = Er(\vec{t} \cdot \vec{d})$$

So

$$Er(\vec{t} \cdot \vec{d}) = S = ||\vec{v}||\theta_t$$

Since $\vec{t}$ is just a unit vector 90 degrees counterclockwise of $\vec{v}$, We can find it by multiplying $\vec{v}$ by the rotation matrix for 90 degrees and normalizing giving

$$\vec{t} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} norm(\vec{v}) = \begin{bmatrix} -norm(\vec{v}).y \\ norm(\vec{v}).x \end{bmatrix}$$

So

$$F_\theta = \frac{\theta_r}{E} = \frac{r(\vec{t} \cdot \vec{d})}{||\vec{v}||} = \frac{r(\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} norm(\vec{v}) \cdot \vec{d})}{||\vec{v}||} = \frac{r(\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \vec{v} \cdot \vec{d})}{||\vec{v}||^2}$$

Why factors

These factors make solving this problem much simpler. For the forward case, for some wheel *i*, its rotation is the sum of all the motions applied to it. So $E_i = F_{xi}x + F_{yi}y + F_{\theta i}\theta$. So, for all the wheels, we plug in the commanded (x, y, θ) to each wheel's factors to get its necessary rotation. Since the factors depend only on the wheels pose in the frame, these can be calculated once at the start of the program and are constant (unless the frame breaks apart, in which case the robot has other problems).

Now Do It Backwards

We have now solved the forward system for when we have a delta of our pose and want our wheel deltas. Now we must take our wheel deltas and solve for our pose delta. We have our formulas for each wheel's encoder motion and can consider this as a system of linear equations. At runtime, we have our wheel encoder deltas we can plug in and then we can solve the system of linear equations. This solving requires that we have enough data to satisfy the equations. That is, we need at least 3 separate wheels with at least some angle between them or else the system will be not fully constrained. In the case of tank odometry, we only have two wheels but as outside observers we know we can not measure change in one dimension. So, we know one variable is zero and then have two remaining free variables and two equations to satisfy the system. For robots with greater than three encoders, we have an over constrained system of equations but this is not an issue. Since all the encoders are modeled on a physical system, they should agree on what the solution is. Using the technique of least squares regression, we can find our (x, y, θ) to solve the over constrained system that minimizes the error between equations. This also gives us a way to detect errors in our drive train. If a wheel gets jammed, its encoder reading will disagree with the rest of the system, and the error value will measurably increase. If we monitor this error value we can diagnose mechanical or electrical issues from the code.

$$T = \begin{bmatrix} F_{x1} & F_{y1} & F_{\theta 1} \\ \vdots & \vdots & \vdots \\ F_{xn} & F_{yn} & F_{\theta n} \end{bmatrix}$$

$$\begin{bmatrix} \frac{\text{Length}}{\text{Angle}} & \frac{\text{Length}}{\text{Angle}} & \frac{\text{Angle}}{\text{Angle}} \\ \vdots & \vdots & \vdots \end{bmatrix}$$

transfer matrix
from robot velocity
to encoder velocities
(f for factor)

$$\vec{X} = \begin{bmatrix} \frac{dx_{robot}}{dt} \\ \frac{dy_{robot}}{dt} \\ \frac{d\theta_{robot}}{dt} \end{bmatrix}$$

$$\begin{bmatrix} \frac{\text{Distance}}{\text{Time}} \\ \frac{\text{Distance}}{\text{Time}} \\ \frac{\text{Angle}}{\text{Time}} \end{bmatrix}$$

pose velocity

$$\vec{E} = \begin{bmatrix} E_1 \\ \vdots \\ E_n \end{bmatrix}$$

$$\begin{bmatrix} \frac{\text{Angle}}{\text{Time}} \\ \vdots \\ \frac{\text{Angle}}{\text{Time}} \end{bmatrix}$$

encoder wheel
velocities

$$T\vec{X} = \vec{E}$$

$$\vec{X} = T^{-1}\vec{E}$$

or in the case where the matrix is not invertible, find the best solution

The linear algebra behind the solution

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

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:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

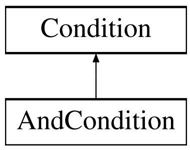
Here is a list of all documented files with brief descriptions:

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:

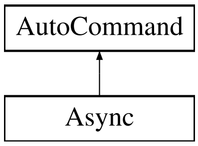
- src/utils/command_structure/auto_command.cpp

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.

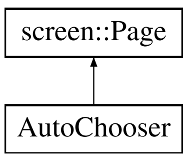
The documentation for this class was generated from the following files:

- 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 autonomi 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

brain	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

first_draw	true if we just switched to this page
frame_number	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

was_pressed	true if the screen has been pressed
x	x position of screen press (if the screen was pressed)
y	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]
```

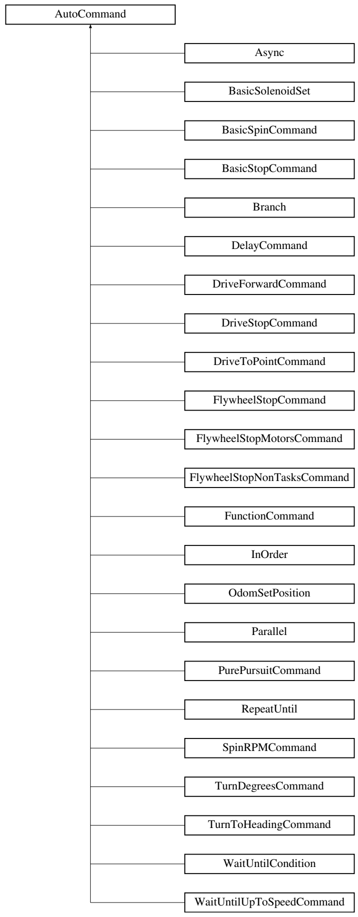
< a list of all possible auto choices

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

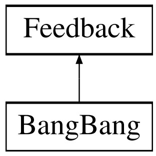
- a drive command hitting a wall and not being able to reach its target
- a command that waits until something is up to speed that never gets up to speed because of battery voltage
- something else...

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 thresshold, 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

start_pt	the current sensor value
set_pt	where the sensor value should be
start_vel	Movement starting velocity
end_vel	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

lower	Upper limit
upper	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

val	value from the sensor

Returns

feedback loop result

Implements Feedback.

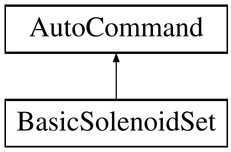
The documentation for this class was generated from the following files:

- 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

solenoid	Solenoid being set
setting	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.

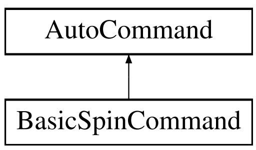
The documentation for this class was generated from the following files:

- 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

motor	Motor to spin
direc	Direction of motor spin
setting	Power setting in volts,percentage,velocity
power	Value of desired power
motor	Motor port to spin
dir	Direction for spining
setting	Power setting in volts,percentage,velocity
power	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.

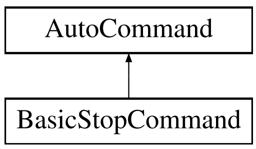
The documentation for this class was generated from the following files:

- 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

motor	The motor to stop
setting	The brake setting for the motor
motor	Motor to stop
setting	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:

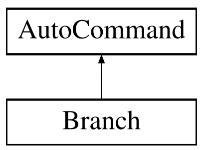
- include/utils/command_structure/basic_command.h
- src/utils/command_structure/basic_command.cpp

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::ButtonConfig Struct Reference

Public Attributes

- std::function< void()> **onclick**

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

- include/subsystems/screen.h

5.11 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

5.11.1 Detailed Description

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

5.11.2 Constructor & Destructor Documentation

5.11.2.1 ButtonWidget() [1/2]

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

Create a Button widget.

Parameters

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

5.11.2.2 ButtonWidget() [2/2]

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

Create a Button widget.

Parameters

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

5.11.3 Member Function Documentation

5.11.3.1 update()

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

responds to user input

Parameters

was_pressed	if the screen is pressed
x	x position if the screen was pressed
y	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

5.12 screen::CheckboxConfig Struct Reference

Public Attributes

- std::function< void(bool)> **onupdate**

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

- include/subsystems/screen.h

5.13 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 * > cmds)

  Create a CommandController with commands pre added. More can be added with CommandController::add()
- void add (std::vector< AutoCommand * > cmds)
- void add (AutoCommand *cmd, double timeout_seconds=10.0)
- void add (std::vector< AutoCommand * > cmds, 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 ()

5.13.1 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.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 CommandController()

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

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

Parameters

cmds	

5.13.3 Member Function Documentation

5.13.3.1 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

cmd	the AutoCommand we want to add to our list
timeout_seconds	the number of seconds we will let the command run for. If it exceeds this, we cancel it and run on_timeout

5.13.3.2 add() [2/3]

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

Adds a command to the queue

Parameters

cmd	the AutoCommand we want to add to our list
timeout_seconds	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

cmds	the AutoCommands we want to add to our list

5.13.3.3 add() [3/3]

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

Add multiple commands to the queue. No timeout here.

Parameters

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

Add multiple commands to the queue. No timeout here.

Parameters

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

5.13.3.4 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

true_if_cancel	a function that returns true when we want to cancel the command controller

5.13.3.5 add_delay()

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

Adds a command that will delay progression of the queue

Parameters

ms	- number of milliseconds to wait before continuing execution of autonomous

5.13.3.6 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

5.13.3.7 run()

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

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

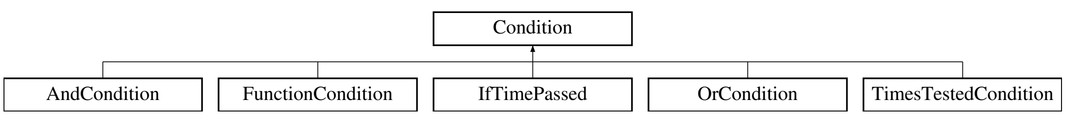
The documentation for this class was generated from the following files:

- include/utils/command_structure/command_controller.h
- src/utils/command_structure/command_controller.cpp

5.14 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

5.14.1 Detailed Description

File: auto_command.h Desc: Interface for module-specifc 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

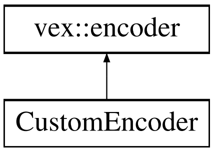
The documentation for this class was generated from the following files:

- include/utils/command_structure/auto_command.h
- src/utils/command_structure/auto_command.cpp

5.15 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)

5.15.1 Detailed Description

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

5.15.2 Constructor & Destructor Documentation

5.15.2.1 CustomEncoder()

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

Construct an encoder with a custom number of ticks

Parameters

port	the triport port on the brain the encoder is plugged into
ticks_per_rev	the number of ticks the encoder will report for one revolution

5.15.3 Member Function Documentation

5.15.3.1 position()

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

get the position that the encoder is at

Parameters

units	the unit we want the return value to be in

Returns

the position of the encoder in the units specified

5.15.3.2 rotation()

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

get the rotation that the encoder is at

Parameters

units	the unit we want the return value to be in

Returns

the rotation of the encoder in the units specified

5.15.3.3 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

val	the numerical value of the position we are setting to
units	the unit of val

5.15.3.4 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

val	the numerical value of the angle we are setting to
units	the unit of val

5.15.3.5 velocity()

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

get the velocity that the encoder is moving at

Parameters

units	the unit we want the return value to be in

Returns

the velocity of the encoder in the units specified

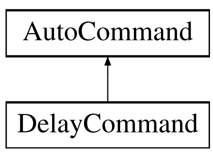
The documentation for this class was generated from the following files:

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

5.16 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

5.16.1 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

5.16.2 Constructor & Destructor Documentation

5.16.2.1 DelayCommand()

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

Construct a delay command

Parameters

ms	the number of milliseconds to delay for

5.16.3 Member Function Documentation

5.16.3.1 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.

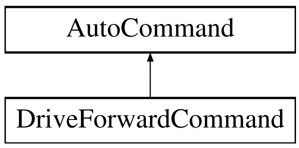
The documentation for this class was generated from the following file:

- include/utils/command_structure/delay_command.h

5.17 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

5.17.1 Detailed Description

AutoCommand wrapper class for the drive_forward function in the TankDrive class

5.17.2 Constructor & Destructor Documentation

5.17.2.1 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

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

5.17.3 Member Function Documentation

5.17.3.1 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.

5.17.3.2 run()

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

Run drive_forward Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

Reimplemented from AutoCommand.

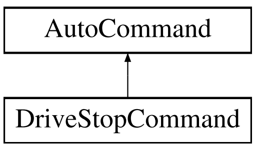
The documentation for this class was generated from the following files:

- include/utils/command_structure/drive_commands.h
- src/utils/command_structure/drive_commands.cpp

5.18 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

5.18.1 Detailed Description

AutoCommand wrapper class for the stop() function in the TankDrive class

5.18.2 Constructor & Destructor Documentation

5.18.2.1 DriveStopCommand()

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

Construct a DriveStop Command

Parameters

drive_sys	the drive system we are commanding

5.18.3 Member Function Documentation

5.18.3.1 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.

5.18.3.2 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.

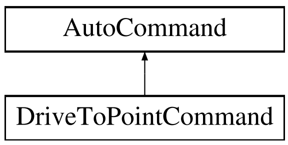
The documentation for this class was generated from the following files:

- include/utils/command_structure/drive_commands.h
- src/utils/command_structure/drive_commands.cpp

5.19 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

5.19.1 Detailed Description

AutoCommand wrapper class for the drive_to_point function in the TankDrive class

5.19.2 Constructor & Destructor Documentation

5.19.2.1 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

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

5.19.2.2 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

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

5.19.3 Member Function Documentation

5.19.3.1 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

5.20 AutoChooser::entry_t Struct Reference

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

Public Attributes

- Rect **rect**
- std::string name

5.20.1 Detailed Description

entry_t is a datatype used to store information that the chooser knows about an auto selection button

5.20.2 Member Data Documentation

5.20.2.1 name

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

name of the auto repretsented by the block

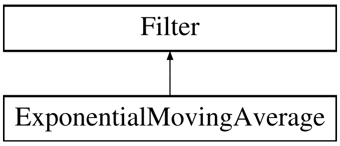
The documentation for this struct was generated from the following file:

- include/utils/auto_chooser.h

5.21 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) override
- double get_value () const override
- int get_size ()

5.21.1 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.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 ExponentialMovingAverage() [1/2]

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

Create a moving average calculator with 0 as the default value

Parameters

buffer_size	The size of the buffer. The number of samples that constitute a valid reading

5.21.2.2 ExponentialMovingAverage() [2/2]

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

Create a moving average calculator with a specified default value

Parameters

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

5.21.3 Member Function Documentation

5.21.3.1 add_entry()

```
void ExponentialMovingAverage::add_entry (
            double n ) [override], [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

n	the sample that will be added to the moving average.

Implements Filter.

5.21.3.2 get_size()

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

How many samples the average is made from

Returns

the number of samples used to calculate this average

5.21.3.3 get_value()

```
double ExponentialMovingAverage::get_value ( ) const [override], [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.

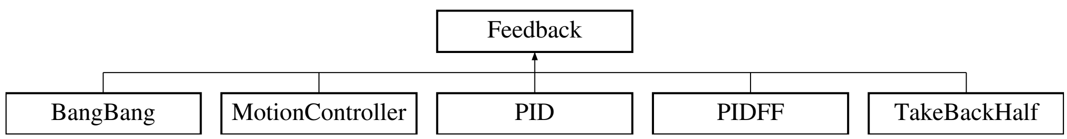
The documentation for this class was generated from the following files:

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

5.22 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

5.22.1 Detailed Description

Interface so that subsystems can easily switch between feedback loops

Author

Ryan McGee

Date

9/25/2022

5.22.2 Member Function Documentation

5.22.2.1 get()

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

Returns

the last saved result from the feedback controller

Implemented in BangBang, MotionController, PID, PIDFF, and TakeBackHalf.

5.22.2.2 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

start_pt	the current sensor value
set_pt	where the sensor value should be
start_vel	Movement starting velocity
end_vel	Movement ending velocity

Implemented in MotionController, PIDFF, PID, BangBang, and TakeBackHalf.

5.22.2.3 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.

5.22.2.4 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

lower	Upper limit
upper	Lower limit

Implemented in BangBang, MotionController, PID, PIDFF, and TakeBackHalf.

5.22.2.5 update()

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

Iterate the feedback loop once with an updated sensor value

Parameters

val	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

5.23 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.

5.23.1 Detailed Description

FeedForward

Stores the feedfoward constants, and allows for quick computation. Feedfoward 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 reccommend 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

5.23.2 Constructor & Destructor Documentation

5.23.2.1 FeedForward()

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

Creates a FeedForward object.

Parameters

cfg	Configuration Struct for tuning

5.23.3 Member Function Documentation

5.23.3.1 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∗sgn(v) + kV∗v + kA∗a

Parameters

v	Requested velocity of system
a	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

5.24 FeedForward::ff_config_t Struct Reference

`#include <feedforward.h>`

Public Attributes

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

5.24.1 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

5.24.2 Member Data Documentation

5.24.2.1 kA

`double FeedForward::ff_config_t::kA`

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

5.24.2.2 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.

5.24.2.3 kS

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

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

5.24.2.4 kV

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

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

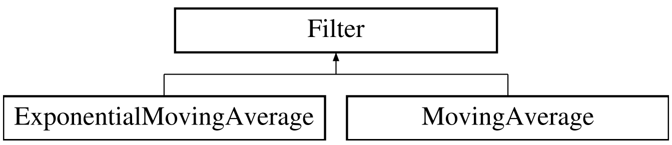
The documentation for this struct was generated from the following file:

- include/utils/controls/feedforward.h

5.25 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

5.25.1 Detailed Description

Interface for filters Use add_entry to supply data and get_value to retrieve the filtered value

5.25.2 Member Function Documentation

5.25.2.1 add_entry()

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

Implemented in MovingAverage, and ExponentialMovingAverage.

5.25.2.2 get_value()

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

Implemented in MovingAverage, and ExponentialMovingAverage.

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

- include/utils/moving_average.h

5.26 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)

5.26.1 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

5.26.2 Constructor & Destructor Documentation

5.26.2.1 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

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

5.26.3 Member Function Documentation

5.26.3.1 get_motors()

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

Returns the motors

Returns

the motors used to run the flywheel

5.26.3.2 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

5.26.3.3 getRPM()

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

return the velocity of the flywheel

5.26.3.4 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

5.26.3.5 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()});

5.26.3.6 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

speed	- speed (between -1 and 1) to set the motor
dir	- 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

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

5.26.3.7 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

rpm	- 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

input_rpm	- set the current RPM

5.26.3.8 SpinRpmCmd()

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

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

Parameters

rpm	the rpm to spin at

Returns

an auto command to add to a command controller

5.26.3.9 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

5.26.3.10 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

5.26.4 Friends And Related Symbol Documentation

5.26.4.1 spinRPMTask

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

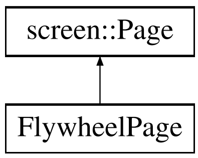
Runs a thread that keeps track of updating flywheel RPM and controlling it accordingly

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

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

5.27 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

5.27.1 Member Function Documentation

5.27.1.1 draw()

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

See also

Page::draw

Reimplemented from screen::Page.

5.27.1.2 update()

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

See also

Page::update

Reimplemented from screen::Page.

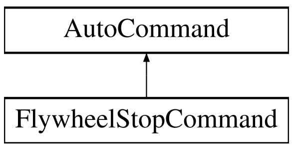
The documentation for this class was generated from the following file:

- src/subsystems/flywheel.cpp

5.28 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

5.28.1 Detailed Description

AutoCommand wrapper class for the stop function in the Flywheel class

5.28.2 Constructor & Destructor Documentation

5.28.2.1 FlywheelStopCommand()

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

Construct a FlywheelStopCommand

Parameters

flywheel	the flywheel system we are commanding

5.28.3 Member Function Documentation

5.28.3.1 run()

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

Run stop Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

Reimplemented from AutoCommand.

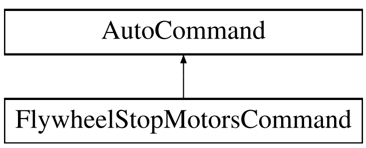
The documentation for this class was generated from the following files:

- include/utils/command_structure/flywheel_commands.h
- src/utils/command_structure/flywheel_commands.cpp

5.29 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

5.29.1 Detailed Description

AutoCommand wrapper class for the stopMotors function in the Flywheel class

5.29.2 Constructor & Destructor Documentation

5.29.2.1 FlywheelStopMotorsCommand()

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

Construct a FlywheeStopMotors Command

Parameters

flywheel	the flywheel system we are commanding

5.29.3 Member Function Documentation

5.29.3.1 run()

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

Run stop Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

Reimplemented from AutoCommand.

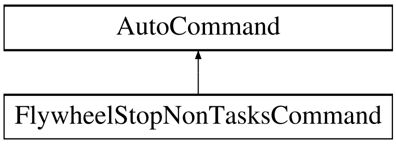
The documentation for this class was generated from the following files:

- include/utils/command_structure/flywheel_commands.h
- src/utils/command_structure/flywheel_commands.cpp

5.30 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

5.30.1 Detailed Description

AutoCommand wrapper class for the stopNonTasks function in the Flywheel class

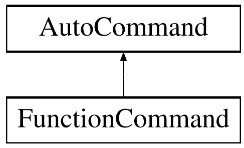
The documentation for this class was generated from the following files:

- include/utils/command_structure/flywheel_commands.h
- src/utils/command_structure/flywheel_commands.cpp

5.31 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

5.31.1 Detailed Description

FunctionCommand is fun and good way to do simple things Printing, launching nukes, and other quick and dirty one time things

5.31.2 Member Function Documentation

5.31.2.1 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:

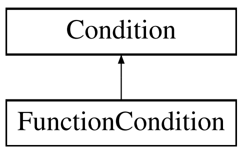
- include/utils/command_structure/auto_command.h

5.32 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)

5.32.1 Detailed Description

FunctionCondition is a quick and dirty Condition to wrap some expression that should be evaluated at runtime.

5.32.2 Member Function Documentation

5.32.2.1 test()

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

Implements Condition.

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

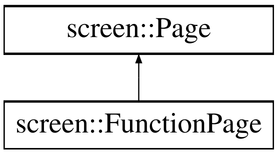
- include/utils/command_structure/auto_command.h
- src/utils/command_structure/auto_command.cpp

5.33 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

5.33.1 Detailed Description

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

5.33.2 Constructor & Destructor Documentation

5.33.2.1 FunctionPage()

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

Creates a function page.

FunctionPage.

Parameters

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

5.33.3 Member Function Documentation

5.33.3.1 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.

5.33.3.2 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

5.34 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)

5.34.1 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

5.34.2 Member Function Documentation

5.34.2.1 add()

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

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

Parameters

new_state	the function to run

5.34.2.2 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

async_state	the function to run

5.34.2.3 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

ms	how long to wait in milliseconds

5.34.2.4 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

blocking	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

5.35 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)

5.35.1 Constructor & Destructor Documentation

5.35.1.1 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

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

5.35.2 Member Function Documentation

5.35.2.1 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

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

5.35.2.2 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

sample	an x, y coordinate of the next point to graph

5.35.2.3 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

x	x position of the top left of the graphed region

Parameters

y	y position of the top left of the graphed region
width	the width of the graphed region
height	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

5.36 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**

5.36.1 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:

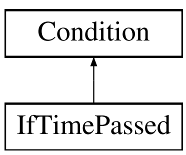
- include/utils/pure_pursuit.h

5.37 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)

5.37.1 Detailed Description

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

5.37.2 Member Function Documentation

5.37.2.1 test()

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

Implements Condition.

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

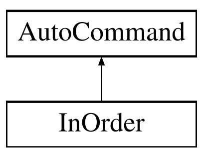
- include/utils/command_structure/auto_command.h
- src/utils/command_structure/auto_command.cpp

5.38 InOrder Class Reference

InOrder runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timouts?

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

Inheritance diagram for InOrder:



Public Member Functions

- **InOrder** (const InOrder &other)=default
- **InOrder** (std::queue< AutoCommand ∗ > cmds)
- **InOrder** (std::initializer_list< AutoCommand ∗ > cmds)
- 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

5.38.1 Detailed Description

InOrder runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timouts?

InOrder runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timouts?

5.38.2 Member Function Documentation

5.38.2.1 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.

5.38.2.2 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

5.39 screen::LabelConfig Struct Reference

Public Attributes

- std::string **label**

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

- include/subsystems/screen.h

5.40 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))

5.40.1 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

5.40.2 Constructor & Destructor Documentation

5.40.2.1 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

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

5.40.3 Member Function Documentation

5.40.3.1 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

up_ctrl	Button controlling the "UP" motion
down_ctrl	Button controlling the "DOWN" motion

5.40.3.2 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

up_btn	Raise the lift when true
down_btn	Lower the lift when true
volt_up	Motor voltage when raising the lift
volt_down	Motor voltage when lowering the lift

5.40.3.3 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

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

5.40.3.4 get_async()

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

Returns

whether or not the background thread is running the lift

5.40.3.5 get_setpoint()

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

Returns

The current setpoint for the lift

5.40.3.6 hold()

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

Target the class's setpoint. Calculate the PID output and set the lift motors accordingly.

5.40.3.7 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.

5.40.3.8 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

val	Whether or not the background thread should run the lift

5.40.3.9 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

pos	A lift position enum type

Returns

True if the pid has reached the setpoint

5.40.3.10 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

fn_ptr	Pointer to custom sensor function

5.40.3.11 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

5.40.3.12 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

val	Lift setpoint, in motor revolutions or sensor units defined by get_sensor. 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

5.41 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**

5.41.1 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

5.42 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 &l)=delete

  copying not allowed
- Logger & **operator=** (const Logger &l)=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

5.42.1 Detailed Description

Class to simplify writing to files.

5.42.2 Constructor & Destructor Documentation

5.42.2.1 Logger()

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

Create a logger that will save to a file.

Parameters

filename	the file to save to

5.42.3 Member Function Documentation

5.42.3.1 Log() [1/2]

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

Write a string to the log.

Parameters

s	the string to write

5.42.3.2 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

5.42.3.3 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

5.42.3.4 Logf() [2/2]

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

Write a formatted string to the log with a loglevel.

Parameters

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

5.42.3.5 Logln() [1/2]

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

Write a string and newline to the log.

Parameters

s	the string to write

5.42.3.6 Logln() [2/2]

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

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

Parameters

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

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

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

5.43 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

5.43.1 Detailed Description

m_profile_config holds all data the motion controller uses to plan paths When motion pofile 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

5.44 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

5.45 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)

5.45.1 Detailed Description

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

5.45.2 Constructor & Destructor Documentation

5.45.2.1 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

5.45.3 Member Function Documentation

5.45.3.1 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

inches	How far the robot should drive, in inches
direction	What direction the robot should travel in, in degrees. 0 is forward, +/-180 is reverse, clockwise is positive.
speed	The maximum speed the robot should travel, in percent: -1.0->+1.0
gyro_correction	=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

inches	How far the robot should drive, in inches
direction	What direction the robot should travel in, in degrees. 0 is forward, +/-180 is reverse, clockwise is positive.
speed	The maximum speed the robot should travel, in percent: -1.0->+1.0
gyro_correction	= 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.

5.45.3.2 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

degrees	How many degrees to rotate the robot. Clockwise postive.
speed	What percentage to run the motors at: 0.0 -> 1.0
ignore_imu	=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

degrees	How many degrees to rotate the robot. Clockwise postive.
speed	What percentage to run the motors at: 0.0 -> 1.0
ignore_imu	= 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

5.45.3.3 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

left_y	left joystick, Y axis (forward / backwards)
left_x	left joystick, X axis (strafe left / right)
right←↩ _x	right joystick, X axis (rotation left / right)
power	=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

left_y	left joystick, Y axis (forward / backwards)
left_x	left joystick, X axis (strafe left / right)
right←↩ _x	right joystick, X axis (rotation left / right)
power	= 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)

5.45.3.4 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

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

5.46 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**

5.46.1 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

5.47 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

5.47.1 Detailed Description

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

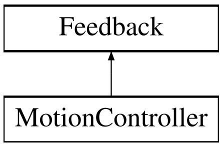
The documentation for this struct was generated from the following file:

- include/utils/controls/trapezoid_profile.h

5.48 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 () const
- screen::Page ∗ **Page** ()

Static Public Member Functions

- static FeedForward::ff_config_t tune_feedforward (TankDrive &drive, OdometryTank &odometry, double pct=0.6, double duration=2)

Friends

- class **MotionControllerPage**

5.48.1 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 = feedfoward.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

5.48.2 Constructor & Destructor Documentation

5.48.2.1 MotionController()

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

Construct a new Motion Controller object.

Parameters

config	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

5.48.3 Member Function Documentation

5.48.3.1 get()

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

Returns

the last saved result from the feedback controller

Implements Feedback.

5.48.3.2 get_motion()

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

Returns

The current postion, velocity and acceleration setpoints

5.48.3.3 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

start_pt	Movement starting position
end_pt	Movement ending posiiton
start_vel	Movement starting velocity
end_vel	Movement ending velocity

Implements Feedback.

5.48.3.4 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.

5.48.3.5 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

lower	upper limit
upper	lower limiet

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

Parameters

lower	Upper limit
upper	Lower limit

Implements Feedback.

5.48.3.6 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

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

Returns

A tuned feedforward object

5.48.3.7 update()

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

Update the motion profile with a new sensor value.

Parameters

sensor_val	Value from the sensor

Returns

the motor input generated from the motion profile

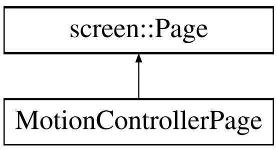
Implements Feedback.

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

- include/utils/controls/motion_controller.h
- src/utils/controls/motion_controller.cpp

5.49 MotionControllerPage Class Reference

Inheritance diagram for MotionControllerPage:



Public Member Functions

- **MotionControllerPage** (const MotionController &mc)
- void update (bool was_pressed, int x, int y) override

  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 &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)

5.49.1 Member Function Documentation

5.49.1.1 draw()

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

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

Parameters

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

Reimplemented from screen::Page.

5.49.1.2 update()

```
void MotionControllerPage::update (
            bool was_pressed,
            int x,
            int y ) [inline], [override], [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

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

Reimplemented from screen::Page.

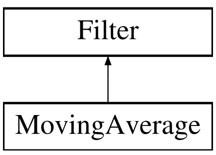
The documentation for this class was generated from the following file:

- src/utils/controls/motion_controller.cpp

5.50 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) override
- double get_value () const override
- int get_size () const

5.50.1 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).

5.50.2 Constructor & Destructor Documentation

5.50.2.1 MovingAverage() [1/2]

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

Create a moving average calculator with 0 as the default value

Parameters

buffer_size	The size of the buffer. The number of samples that constitute a valid reading

5.50.2.2 MovingAverage() [2/2]

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

Create a moving average calculator with a specified default value

Parameters

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

5.50.3 Member Function Documentation

5.50.3.1 add_entry()

```
void MovingAverage::add_entry (
            double n )  [override], [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

n	the sample that will be added to the moving average.

Implements Filter.

5.50.3.2 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

5.50.3.3 get_value()

```
double MovingAverage::get_value ( ) const  [override], [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.

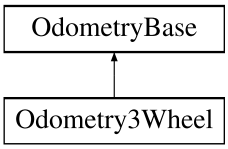
The documentation for this class was generated from the following files:

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

5.51 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)
- AutoCommand ∗ **SetPositionCmd** (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

5.51.1 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:

+Y -------------— ^ | | | | | | | | || O || | | | | | | | === | | -------------— | +-------------------> + X

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

5.51.2 Constructor & Destructor Documentation

5.51.2.1 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

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

5.51.3 Member Function Documentation

5.51.3.1 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

con	Controller reference, for screen and button control
drive	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

5.51.3.2 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

5.52 Odometry3Wheel::odometry3wheel_cfg_t Struct Reference

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

Public Attributes

- double wheelbase_dist
- double off_axis_center_dist
- double wheel_diam

5.52.1 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

5.52.2 Member Data Documentation

5.52.2.1 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

5.52.2.2 wheel_diam

```
double Odometry3Wheel::odometry3wheel_cfg_t::wheel_diam
```

the diameter of the tracking wheel

5.52.2.3 wheelbase_dist

```
double Odometry3Wheel::odometry3wheel_cfg_t::wheelbase_dist
```

distance from the center of the left wheel to the center of the right wheel

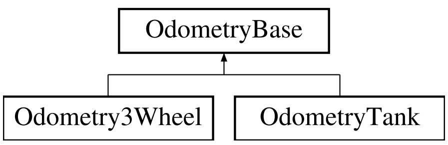
The documentation for this struct was generated from the following file:

- include/subsystems/odometry/odometry_3wheel.h

5.53 OdometryBase Class Reference

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

Inheritance diagram for OdometryBase:



Public Member Functions

- OdometryBase (bool is_async)
- pose_t get_position (void)
- virtual void set_position (const pose_t &newpos=zero_pos)
- AutoCommand ∗ **SetPositionCmd** (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

5.53.1 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

5.53.2 Constructor & Destructor Documentation

5.53.2.1 OdometryBase()

```
OdometryBase::OdometryBase (
            bool is_async )
```

Construct a new Odometry Base object

Parameters

is_async	True to run constantly in the background, false to call update() manually

5.53.3 Member Function Documentation

5.53.3.1 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

ptr	Pointer to OdometryBase object

Returns

Required integer return code. Unused.

5.53.3.2 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.

5.53.3.3 get_accel()

```
double OdometryBase::get_accel ( )
```

Get the current acceleration

Returns

the acceleration rate of the robot (inch/s^2)

5.53.3.4 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^2)

5.53.3.5 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)

5.53.3.6 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

5.53.3.7 get_speed()

```
double OdometryBase::get_speed ( )
```

Get the current speed

Returns

the speed at which the robot is moving and grooving (inch/s)

5.53.3.8 pos_diff()

```
double OdometryBase::pos_diff (
            pose_t start_pos,
            pose_t end_pos ) [static]
```

Get the distance between two points

Parameters

start_pos	distance from this point
end_pos	to this point

Returns

the euclidean distance between start_pos and end_pos

5.53.3.9 rot_diff()

```
double OdometryBase::rot_diff (
            pose_t pos1,
            pose_t pos2 )  [static]
```

Get the change in rotation between two points

Parameters

pos1	position with initial rotation
pos2	position with final rotation

Returns

change in rotation between pos1 and pos2

Get the change in rotation between two points

5.53.3.10 set_position()

```
void OdometryBase::set_position (
            const pose_t & newpos = zero_pos )  [virtual]
```

Sets the current position of the robot

Parameters

newpos	the new position that the odometry will believe it is at

Sets the current position of the robot

Reimplemented in OdometryTank.

5.53.3.11 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

start_deg	intitial angle (degrees)
end_deg	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.

5.53.3.12 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.

5.53.4 Member Data Documentation

5.53.4.1 accel

```
double OdometryBase::accel  [protected]
```

the rate at which we are accelerating (inch/s^2)

5.53.4.2 ang_accel_deg

```
double OdometryBase::ang_accel_deg  [protected]
```

the rate at which we are accelerating our turn (deg/s^2)

5.53.4.3 ang_speed_deg

```
double OdometryBase::ang_speed_deg  [protected]
```

the speed at which we are turning (deg/s)

5.53.4.4 current_pos

```
pose_t OdometryBase::current_pos [protected]
```

Current position of the robot in terms of x,y,rotation

5.53.4.5 handle

```
vex::task* OdometryBase::handle [protected]
```

handle to the vex task that is running the odometry code

5.53.4.6 mut

```
vex::mutex OdometryBase::mut [protected]
```

Mutex to control multithreading

5.53.4.7 speed

```
double OdometryBase::speed [protected]
```

the speed at which we are travelling (inch/s)

5.53.4.8 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:

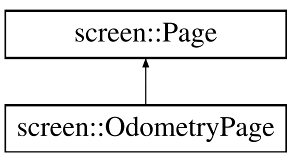
- include/subsystems/odometry/odometry_base.h
- src/subsystems/odometry/odometry_base.cpp

5.54 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 initilized 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

5.54.1 Detailed Description

a page that shows odometry position and rotation and a map (if an sd card with the file is on)

5.54.2 Constructor & Destructor Documentation

5.54.2.1 OdometryPage()

```
screen::OdometryPage::OdometryPage (
            OdometryBase & odom,
            double robot_width,
            double robot_height,
            bool do_trail )
```

Create an odometry trail. Make sure odometry is initilized before now.

Parameters

odom	the odometry system to monitor
robot_width	the width (side to side) of the robot in inches. Used for visualization
robot_height	the robot_height (front to back) of the robot in inches. Used for visualization
do_trail	whether or not to calculate and draw the trail. Drawing and storing takes a very *slight* extra amount of processing power

5.54.3 Member Function Documentation

5.54.3.1 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.

5.54.3.2 update()

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

See also

Page::update

Reimplemented from screen::Page.

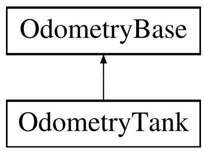
The documentation for this class was generated from the following files:

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

5.55 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)
- AutoCommand ∗ **SetPositionCmd** (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

5.55.1 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.

5.55.2 Constructor & Destructor Documentation

5.55.2.1 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

left_side	The left motors
right_side	The right motors
config	the specifications that supply the odometry with descriptions of the robot. See robot_specs_t for what is contained
imu	The robot's inertial sensor. If not included, rotation is calculated from the encoders.
is_async	If true, position will be updated in the background continuously. If false, the programmer will have to manually call update().

5.55.2.2 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

left_custom_enc	The left custom encoder
right_custom_enc	The right custom encoder
config	the specifications that supply the odometry with descriptions of the robot. See robot_specs_t for what is contained
imu	The robot's inertial sensor. If not included, rotation is calculated from the encoders.
is_async	If true, position will be updated in the background continuously. If false, the programmer will have to manually call update().

5.55.2.3 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

left_vex_enc	The left vex encoder
right_vex_enc	The right vex encoder
config	the specifications that supply the odometry with descriptions of the robot. See robot_specs_t for what is contained
imu	The robot's inertial sensor. If not included, rotation is calculated from the encoders.
is_async	If true, position will be updated in the background continuously. If false, the programmer will have to manually call update().

5.55.3 Member Function Documentation

5.55.3.1 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

newpos	the position the odometry will take

Resets the position and rotational data to the input.

Reimplemented from OdometryBase.

5.55.3.2 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.

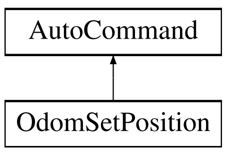
The documentation for this class was generated from the following files:

- include/subsystems/odometry/odometry_tank.h
- src/subsystems/odometry/odometry_tank.cpp

5.56 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

5.56.1 Detailed Description

AutoCommand wrapper class for the set_position function in the Odometry class

5.56.2 Constructor & Destructor Documentation

5.56.2.1 OdomSetPosition()

```
OdomSetPosition::OdomSetPosition (
            OdometryBase & odom,
            const pose_t & newpos = OdometryBase::zero_pos )
```

constructs a new OdomSetPosition command

Parameters

odom	the odometry system we are setting
newpos	the position we are telling the odometry to take. defaults to (0, 0), angle = 90

Construct an Odometry set pos

Parameters

odom	the odometry system we are setting
newpos	the now position to set the odometry to

5.56.3 Member Function Documentation

5.56.3.1 run()

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

Run set_position Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

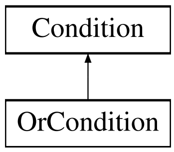
Reimplemented from AutoCommand.

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

5.57 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)

5.57.1 Member Function Documentation

5.57.1.1 test()

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

Implements Condition.

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

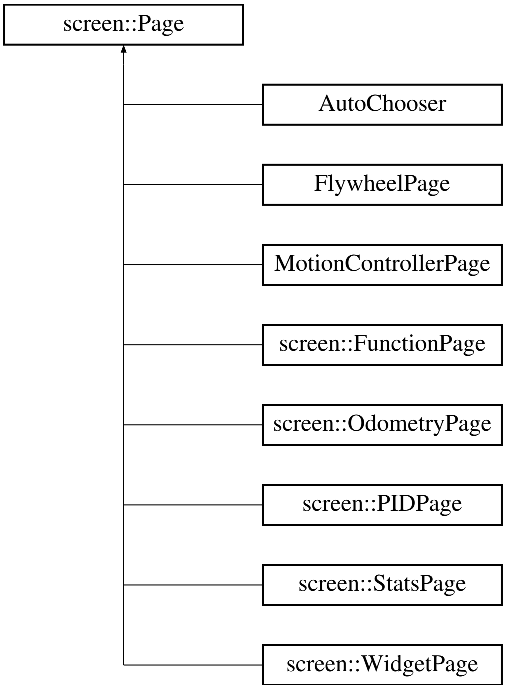
- src/utils/command_structure/auto_command.cpp

5.58 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)

5.58.1 Detailed Description

Page describes one part of the screen slideshow.

5.58.2 Member Function Documentation

5.58.2.1 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

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

Reimplemented in AutoChooser, screen::WidgetPage, screen::StatsPage, screen::OdometryPage, screen::FunctionPage, screen::PIDPage, MotionControllerPage, and FlywheelPage.

5.58.2.2 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

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

Reimplemented in AutoChooser, screen::WidgetPage, screen::StatsPage, screen::OdometryPage, screen::FunctionPage, screen::PIDPage, MotionControllerPage, and FlywheelPage.

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

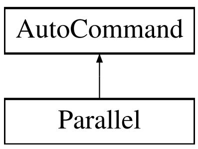
- include/subsystems/screen.h

5.59 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 * > cmds)
- 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

5.59.1 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.

5.59.2 Member Function Documentation

5.59.2.1 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.

5.59.2.2 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

5.60 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

5.61 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 ()

5.61.1 Detailed Description

Wrapper for a vector of points, checking if any of the points are too close for pure pursuit

5.61.2 Constructor & Destructor Documentation

5.61.2.1 Path()

```
PurePursuit::Path::Path (
            std::vector< point_t > points,
            double radius )
```

Create a Path

Parameters

points	the points that make up the path
radius	the lookahead radius for pure pursuit

5.61.3 Member Function Documentation

5.61.3.1 get_points()

```
std::vector< point_t > PurePursuit::Path::get_points ( )
```

Get the points associated with this Path

5.61.3.2 get_radius()

```
double PurePursuit::Path::get_radius ( )
```

Get the radius associated with this Path

5.61.3.3 is_valid()

```
bool PurePursuit::Path::is_valid ( )
```

Get whether this path will behave as expected

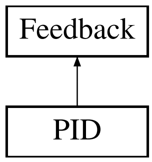
The documentation for this class was generated from the following files:

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

5.62 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 () const

    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 () const
- void set_target (double target)

Public Attributes

- pid_config_t & config

5.62.1 Detailed Description

PID Class

Defines a standard feedback loop using the constants kP, kI, kD, deadband, and on_target_time. The formula is:

out = kP∗error + kI∗integral(d Error) + kD∗(dError/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

5.62.2 Member Enumeration Documentation

5.62.2.1 ERROR_TYPE

```
enum PID::ERROR_TYPE
```

An enum to distinguish between a linear and angular caluclation of PID error.

5.62.3 Constructor & Destructor Documentation

5.62.3.1 PID()

```
PID::PID (
            pid_config_t & config )
```

Create the PID object

Parameters

config	the configuration data for this controller

Create the PID object

5.62.4 Member Function Documentation

5.62.4.1 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.

5.62.4.2 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

5.62.4.3 get_sensor_val()

```
double PID::get_sensor_val ( ) const
```

gets the sensor value that we were last updated with

Returns

sensor_val

5.62.4.4 get_target()

```
double PID::get_target ( ) const
```

Get the PID's target

Returns

the target the PID controller is trying to achieve

5.62.4.5 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

start_pt	commpletely ignored for PID. necessary to satisfy Feedback base
set_pt	sets the target of the PID controller
start_vel	completely ignored for PID. necessary to satisfy Feedback base
end_vel	sets the target end velocity of the PID controller

Implements Feedback.

5.62.4.6 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.

5.62.4.7 reset()

```
void PID::reset ( )
```

Reset the PID loop by resetting time since 0 and accumulated error.

5.62.4.8 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

lower	the lower limit. the PID controller will never command the output go below `lower`
upper	the upper limit. the PID controller will never command the output go higher than `upper`

Set the limits on the PID out. The PID out will "clip" itself to be between the limits.

Implements Feedback.

5.62.4.9 set_target()

```
void PID::set_target (
            double target )
```

Set the target for the PID loop, where the robot is trying to end up

Parameters

target	the sensor reading we would like to achieve

Set the target for the PID loop, where the robot is trying to end up

5.62.4.10 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

sensor_val	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.

5.62.5 Member Data Documentation

5.62.5.1 config

```
pid_config_t& PID::config
```

configuration struct for this controller. see pid_config_t for information about what this contains

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

- include/utils/controls/pid.h
- src/utils/controls/pid.cpp

5.63 PID::pid_config_t Struct Reference

```
#include <pid.h>
```

Public Attributes

- double **p**

  proportional coeffecient p $*$ error()
- double **i**

  integral coeffecient i $*$ integral(error)
- double **d**

  derivitave coeffecient d $*$ derivative(error)
- double **deadband**

  at what threshold are we close enough to be finished
- double on_target_time
- ERROR_TYPE error_method

5.63.1 Detailed Description

pid_config_t holds the configuration parameters for a pid controller In addtion 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

5.63.2 Member Data Documentation

5.63.2.1 error_method

```
ERROR_TYPE PID::pid_config_t::error_method
```

Linear or angular. wheter to do error as a simple subtraction or to wrap

5.63.2.2 on_target_time

`double PID::pid_config_t::on_target_time`

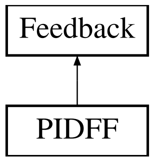
the time in seconds that we have to be on target for to say we are officially at the target

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

- include/utils/controls/pid.h

5.64 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 **get_target** () const
- double **get_sensor_val** () const
- 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
- void **reset** ()

Public Attributes

- PID **pid**

5.64.1 Member Function Documentation

5.64.1.1 get()

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

Returns

the last saved result from the feedback controller

Implements Feedback.

5.64.1.2 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

start_pt	the current sensor value
set_pt	where the sensor value should be
start_vel	the current rate of change of the sensor value
end_vel	the desired ending rate of change of the sensor value

Initialize the feedback controller for a movement

Parameters

start← _pt	the current sensor value
set_pt	where the sensor value should be

Implements Feedback.

5.64.1.3 is_on_target()

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

Returns

true if the feedback controller has reached it's setpoint

Implements Feedback.

5.64.1.4 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

lower	Upper limit
upper	Lower limit

Implements Feedback.

5.64.1.5 set_target()

```
void PIDFF::set_target (
            double set_pt )
```

Set the target of the PID loop

Parameters

set←_pt	Setpoint / target value

5.64.1.6 update() [1/2]

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

Iterate the feedback loop once with an updated sensor value. Only kS for feedfoward will be applied.

Parameters

val	value from the sensor

Returns

feedback loop result

Implements Feedback.

5.64.1.7 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

val	value from the sensor
vel_setpt	Velocity for feedforward
a_setpt	Acceleration for feedfoward

Returns

feedback loop result

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

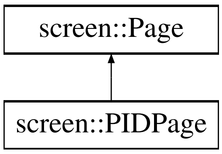
- include/utils/controls/pidff.h
- src/utils/controls/pidff.cpp

5.65 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

5.65.1 Detailed Description

PIDPage provides a way to tune a pid controller on the screen.

5.65.2 Constructor & Destructor Documentation

5.65.2.1 PIDPage()

```
screen::PIDPage::PIDPage (
            PID & pid,
            std::string name,
            std::function< void(void)> onchange = []() {} )
```

Create a PIDPage.

Parameters

pid	the pid controller we're changing
name	a name to recognize this pid controller if we've got multiple pid screens
onchange	a function that is called when a tuning parameter is changed. If you need to update stuff on that change register a handler here

5.65.3 Member Function Documentation

5.65.3.1 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.

5.65.3.2 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

5.66 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

5.66.1 Detailed Description

Data structure representing an X,Y coordinate

5.66.2 Member Function Documentation

5.66.2.1 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

other	the point to measure the distance from

Returns

the euclidian distance between this and other

5.66.2.2 operator+()

```
point_t point_t::operator+ (
            const point_t & other ) const  [inline]
```

Vector2D addition operation on points

Parameters

other	the point to add on to this

Returns

this + other (this.x + other.x, this.y + other.y)

5.66.2.3 operator-()

```
point_t point_t::operator- (
            const point_t & other ) const  [inline]
```

Vector2D subtraction operation on points

Parameters

other	the point_t 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

5.67 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

5.67.1 Detailed Description

Describes a single position and rotation

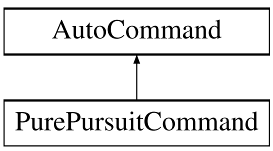
The documentation for this struct was generated from the following file:

- include/utils/geometry.h

5.68 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

5.68.1 Detailed Description

Autocommand wrapper class for pure pursuit function in the TankDrive class

5.68.2 Constructor & Destructor Documentation

5.68.2.1 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

path	The list of coordinates to follow, in order
dir	Run the bot forwards or backwards
feedback	The feedback controller determining speed
max_speed	Limit the speed of the robot (for pid / pidff feedbacks)

5.68.3 Member Function Documentation

5.68.3.1 on_timeout()

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

Reset the drive system when it times out

Reimplemented from AutoCommand.

5.68.3.2 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

5.69 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

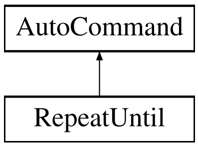
- point_t **min**
- point_t **max**

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

- include/utils/geometry.h

5.70 RepeatUntil Class Reference

Inheritance diagram for RepeatUntil:



Public Member Functions

- RepeatUntil (InOrder cmds, size_t repeats)
    RepeatUntil that runs a fixed number of times.
- RepeatUntil (InOrder cmds, 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

5.70.1 Constructor & Destructor Documentation

5.70.1.1 RepeatUntil() [1/2]

```
RepeatUntil::RepeatUntil (
            InOrder cmds,
            size_t repeats )
```

RepeatUntil that runs a fixed number of times.

Parameters

cmds	the cmds to repeat
repeats	the number of repeats to do

5.70.1.2 RepeatUntil() [2/2]

```
RepeatUntil::RepeatUntil (
            InOrder cmds,
            Condition * true_to_end )
```

RepeatUntil the condition.

Parameters

cmds	the cmds to run
true_to_end	we will repeat until true_or_end.test() returns true

5.70.2 Member Function Documentation

5.70.2.1 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.

5.70.2.2 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

5.71 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

5.71.1 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

5.72 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**

5.72.1 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

5.73 screen::ScreenRect Struct Reference

Public Attributes

- uint32_t **x1**
- uint32_t **y1**
- uint32_t **x2**
- uint32_t **y2**

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

- include/subsystems/screen.h

5.74 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

5.74.1 Detailed Description

Serializes Arbitrary data to a file on the SD Card.

5.74.2 Constructor & Destructor Documentation

5.74.2.1 Serializer()

```
Serializer::Serializer (
            const std::string & filename,
            bool flush_always = true )  [inline], [explicit]
```

create a Serializer

Parameters

filename	the file to read from. If filename does not exist we will create that file
flush_always	If true, after every write flush to a file. If false, you are responsible for calling save_to_disk

5.74.3 Member Function Documentation

5.74.3.1 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

name	name of value
otherwise	value if the name is not specified

Returns

the value if found or otherwise

5.74.3.2 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

name	name of value
otherwise	value if the name is not specified

Returns

the value if found or otherwise

5.74.3.3 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

name	name of value
otherwise	value if the name is not specified

Returns

the value if found or otherwise

5.74.3.4 save_to_disk()

```
void Serializer::save_to_disk ( ) const
```

saves current Serializer state to disk

forms data bytes then saves to filename this was openned with

5.74.3.5 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

name	name of bool
b	value of bool

5.74.3.6 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

name	name of double
d	value of double

5.74.3.7 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

name	name of integer
i	value of integer

5.74.3.8 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

name	name of string
i	value of string

5.74.3.9 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

name	name of value
otherwise	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

5.75 screen::SizedWidget Struct Reference

Public Attributes

- int **size**
- WidgetConfig & **widget**

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

- include/subsystems/screen.h

5.76 SliderCfg Struct Reference

Public Attributes

- double & **val**
- double **min**
- double **max**

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

- include/subsystems/layout.h

5.77 screen::SliderConfig Struct Reference

Public Attributes

- double & **val**
- double **low**
- double **high**

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

- include/subsystems/screen.h

5.78 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

5.78.1 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.

5.78.2 Constructor & Destructor Documentation

5.78.2.1 SliderWidget()

```
screen::SliderWidget::SliderWidget (
            double & val,
            double low,
            double high,
            Rect rect,
            std::string name )  [inline]
```

Creates a slider widget.

Parameters

val	reference to the value to modify
low	minimum value to go to
high	maximum value to go to
rect	rect to draw it
name	name of the value

5.78.3 Member Function Documentation

5.78.3.1 update()

```
bool screen::SliderWidget::update (
```

```
bool was_pressed,
int x,
int y )
```

responds to user input

Parameters

was_pressed	if the screen is pressed
x	x position if the screen was pressed
y	y position if the screen was pressed

Returns

true if the value updated

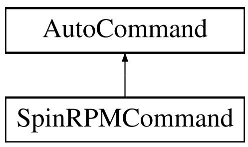
The documentation for this class was generated from the following files:

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

5.79 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

5.79.1 Detailed Description

File: flywheel_commands.h Desc: [insert meaningful desc] AutoCommand wrapper class for the spin_rpm function in the Flywheel class

5.79.2 Constructor & Destructor Documentation

5.79.2.1 SpinRPMCommand()

```
SpinRPMCommand::SpinRPMCommand (
            Flywheel & flywheel,
            int rpm )
```

Construct a SpinRPM Command

Parameters

flywheel	the flywheel sys to command
rpm	the rpm that we should spin at

File: flywheel_commands.cpp Desc: [insert meaningful desc]

5.79.3 Member Function Documentation

5.79.3.1 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

5.80 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**

5.80.1 Detailed Description

Represents a piece of a cubic spline with s(x) = a(x-xi)^3 + b(x-xi)^2 + c(x-xi) + d The x_start and x_end shows where the equation is valid.

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

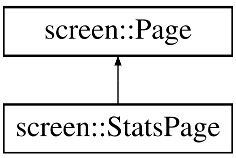
- include/utils/pure_pursuit.h

5.81 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

5.81.1 Detailed Description

Draws motor stats and battery stats to the screen.

5.81.2 Constructor & Destructor Documentation

5.81.2.1 StatsPage()

```
screen::StatsPage::StatsPage (
            std::map< std::string, vex::motor & > motors )
```

Creates a stats page.

Parameters

motors	a map of string to motor that we want to draw on this page

5.81.3 Member Function Documentation

5.81.3.1 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.

5.81.3.2 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:

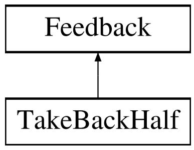
- include/subsystems/screen.h
- src/subsystems/screen.cpp

5.82 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**

5.82.1 Detailed Description

A velocity controller.

Warning

If you try to use this as a position controller, it will fail.

5.82.2 Member Function Documentation

5.82.2.1 get()

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

Returns

the last saved result from the feedback controller

Implements Feedback.

5.82.2.2 init()

```
void TakeBackHalf::init (
            double start_pt,
            double set_pt,
            double ,
            double )  [virtual]
```

Initialize the feedback controller for a movement

Parameters

start_pt	the current sensor value
set_pt	where the sensor value should be
start_vel	Movement starting velocity (IGNORED)
end_vel	Movement ending velocity (IGNORED)

Implements Feedback.

5.82.2.3 is_on_target()

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

Returns

true if the feedback controller has reached it's setpoint

Implements Feedback.

5.82.2.4 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

lower	Upper limit
upper	Lower limit

Implements Feedback.

5.82.2.5 update()

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

Iterate the feedback loop once with an updated sensor value

Parameters

val	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

5.83 TankDrive Class Reference

```
#include <tank_drive.h>
```

Public Types

- enum class BrakeType { None , ZeroVelocity , Smart }

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, BrakeType bt=BrakeType::None)
- void drive_tank_raw (double left, double right)
- void drive_arcade (double forward_back, double left_right, int power=1, BrakeType bt=BrakeType::None)
- 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)

5.83.1 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

5.83.2 Member Enumeration Documentation

5.83.2.1 BrakeType

```
enum class TankDrive::BrakeType  [strong]
```

Enumerator

None	just send 0 volts to the motors
ZeroVelocity	try to bring the robot to rest. But don't try to hold position
Smart	bring the robot to rest and once it's stopped, try to hold that position

5.83.3 Constructor & Destructor Documentation

5.83.3.1 TankDrive()

```
TankDrive::TankDrive (
            motor_group & left_motors,
            motor_group & right_motors,
            robot_specs_t & config,
            OdometryBase * odom = NULL )
```

Create the TankDrive object

Parameters

left_motors	left side drive motors

Parameters

right_motors	right side drive motors
config	the configuration specification defining physical dimensions about the robot. See robot_specs_t for more info
odom	an odometry system to track position and rotation. this is necessary to execute autonomous paths

5.83.4 Member Function Documentation

5.83.4.1 drive_arcade()

```
void TankDrive::drive_arcade (
            double forward_back,
            double left_right,
            int power = 1,
            BrakeType bt = BrakeType::None )
```

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

forward_back	the percent to move forward or backward
left_right	the percent to turn left or right
power	modifies the input velocities left^power, right^power
bt	breaktype. What to do if the driver lets go of the sticks

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

5.83.4.2 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

inches	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
dir	the direction we want to travel forward and backward
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Autonomously drive the robot forward a certain distance

Parameters

inches	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
dir	the direction we want to travel forward and backward
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have finished driving to our point

5.83.4.3 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

inches	the distance to drive forward
dir	the direction we want to travel forward and backward
feedback	the custom feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	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

inches	the distance to drive forward
dir	the direction we want to travel forward and backward
feedback	the custom feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

5.83.4.4 drive_tank()

```
void TankDrive::drive_tank (
            double left,
            double right,
            int power = 1,
            BrakeType bt = BrakeType::None )
```

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

left	the percent to run the left motors
right	the percent to run the right motors
power	modifies the input velocities left^power, right^power
bt	breaktype. What to do if the driver lets go of the sticks

5.83.4.5 drive_tank_raw()

```
void TankDrive::drive_tank_raw (
            double left,
            double right )
```

Drive the robot raw-ly

Parameters

left	the percent to run the left motors (-1, 1)
right	the percent to run the right motors (-1, 1)

5.83.4.6 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

x	the x position of the target

Parameters

y	the y position of the target
dir	the direction we want to travel forward and backward
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	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

x	the x position of the target
y	the y position of the target
dir	the direction we want to travel forward and backward
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have reached our target point

5.83.4.7 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

x	the x position of the target
y	the y position of the target
dir	the direction we want to travel forward and backward
feedback	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	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

x	the x position of the target
y	the y position of the target
dir	the direction we want to travel forward and backward
feedback	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have reached our target point

5.83.4.8 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

input	the input before modification
power	the power to raise input to

Returns

input ^ 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

input	the input signal -1 -> 1
power	the power to raise the signal to

Returns

input^power accounting for any sign issues that would arise with this naive solution

5.83.4.9 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

path	The list of coordinates to follow, in order
dir	Run the bot forwards or backwards
max_speed	Limit the speed of the robot (for pid / pidff feedbacks)
end_speed	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

path	The list of coordinates to follow, in order
dir	Run the bot forwards or backwards
max_speed	Limit the speed of the robot (for pid / pidff feedbacks)

Returns

True when the path is complete

5.83.4.10 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

path	The list of coordinates to follow, in order
dir	Run the bot forwards or backwards
feedback	The feedback controller determining speed
max_speed	Limit the speed of the robot (for pid / pidff feedbacks)
end_speed	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

path	The list of coordinates to follow, in order
dir	Run the bot forwards or backwards
feedback	The feedback controller determining speed
max_speed	Limit the speed of the robot (for pid / pidff feedbacks)

Returns

True when the path is complete

5.83.4.11 reset_auto()

```
void TankDrive::reset_auto ( )
```

Reset the initialization for autonomous drive functions

5.83.4.12 stop()

```
void TankDrive::stop ( )
```

Stops rotation of all the motors using their "brake mode"

5.83.4.13 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 defualt turning feedback of the drive system.

Parameters

degrees	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	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 defualt turning feedback of the drive system.

Parameters

degrees	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

true if we turned te target number of degrees

5.83.4.14 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 it's control.

Parameters

degrees	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
feedback	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	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 it's control.

Parameters

degrees	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
feedback	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have turned our target number of degrees

5.83.4.15 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

heading_deg	the heading to which we will turn
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	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

heading_deg	the heading to which we will turn
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have reached our target heading

5.83.4.16 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

heading_deg	the heading to which we will turn
feedback	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	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

heading_deg	the heading to which we will turn
feedback	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	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

5.84 screen::TextConfig Struct Reference

Public Attributes

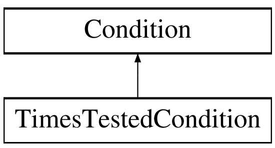
- std::function< std::string()> **text**

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

- include/subsystems/screen.h

5.85 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)

5.85.1 Member Function Documentation

5.85.1.1 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

5.86 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

5.86.1 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

5.87 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 () const

  uses the kinematic equations to and specified accel and max_v to figure out how long moving along the profile would take
- double **get_max_v** () const
- double **get_accel** () const

5.87.1 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 feedfoward 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/rkm3ivu1yk

Author

Ryan McGee

Date

7/12/2022

5.87.2 Constructor & Destructor Documentation

5.87.2.1 TrapezoidProfile()

```
TrapezoidProfile::TrapezoidProfile (
            double max_v,
            double accel )
```

Construct a new Trapezoid Profile object.

Parameters

max_v	Maximum velocity the robot can run at
accel	Maximum acceleration of the robot

5.87.3 Member Function Documentation

5.87.3.1 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

time_s	Time since start of movement
pos_s	The current position

Returns

motion_t Position, velocity and acceleration

5.87.3.2 calculate_time_based()

```
motion_t TrapezoidProfile::calculate_time_based (
            double time_s )
```

Run the trapezoidal profile based on the time that's elapsed.

Parameters

time_s	Time since start of movement

Returns

motion_t Position, velocity and acceleration

5.87.3.3 get_movement_time()

```
double TrapezoidProfile::get_movement_time ( ) const
```

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

5.87.3.4 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

accel	the acceleration amount to use

5.87.3.5 set_endpts()

```
void TrapezoidProfile::set_endpts (
            double start,
            double end )
```

set_endpts defines a start and end position

Parameters

start	the starting position of the path
end	the ending position of the path

5.87.3.6 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

max_v	the maximum velocity the robot can travel at

5.87.3.7 set_vel_endpts()

```
void TrapezoidProfile::set_vel_endpts (
            double start,
            double end )
```

set start and end velocities

Parameters

start	the starting velocity of the path
end	the ending velocity of the path

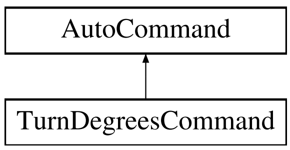
The documentation for this class was generated from the following files:

- include/utils/controls/trapezoid_profile.h
- src/utils/trapezoid_profile.cpp

5.88 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

5.88.1 Detailed Description

AutoCommand wrapper class for the turn_degrees function in the TankDrive class

5.88.2 Constructor & Destructor Documentation

5.88.2.1 TurnDegreesCommand()

```
TurnDegreesCommand::TurnDegreesCommand (
            TankDrive & drive_sys,
            Feedback & feedback,
            double degrees,
            double max_speed = 1,
            double end_speed = 0 )
```

Construct a TurnDegreesCommand Command

Parameters

drive_sys	the drive system we are commanding
feedback	the feedback controller we are using to execute the turn
degrees	how many degrees to rotate
max_speed	0 -> 1 percentage of the drive systems speed to drive at

5.88.3 Member Function Documentation

5.88.3.1 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.

5.88.3.2 run()

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

Run turn_degrees Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

Reimplemented from AutoCommand.

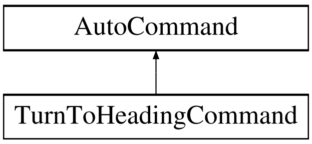
The documentation for this class was generated from the following files:

- include/utils/command_structure/drive_commands.h
- src/utils/command_structure/drive_commands.cpp

5.89 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

5.89.1 Detailed Description

AutoCommand wrapper class for the turn_to_heading() function in the TankDrive class

5.89.2 Constructor & Destructor Documentation

5.89.2.1 TurnToHeadingCommand()

```
TurnToHeadingCommand::TurnToHeadingCommand (
            TankDrive & drive_sys,
            Feedback & feedback,
            double heading_deg,
            double max_speed = 1,
            double end_speed = 0 )
```

Construct a TurnToHeadingCommand Command

Parameters

drive_sys	the drive system we are commanding
feedback	the feedback controller we are using to execute the drive
heading_deg	the heading to turn to in degrees
max_speed	0 -> 1 percentage of the drive systems speed to drive at

5.89.3 Member Function Documentation

5.89.3.1 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.

5.89.3.2 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

5.90 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)

5.90.1 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

5.90.2 Constructor & Destructor Documentation

5.90.2.1 Vector2D() [1/2]

```
Vector2D::Vector2D (
            double dir,
            double mag )
```

Construct a vector object.

Parameters

dir	Direction, in radians. 'foward' is 0, clockwise positive when viewed from the top.
mag	Magnitude.

5.90.2.2 Vector2D() [2/2]

```
Vector2D::Vector2D (
            point_t p )
```

Construct a vector object from a cartesian point.

Parameters

p	point_t.x , point_t.y

5.90.3 Member Function Documentation

5.90.3.1 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 vetctor in radians

Get the direction of the vector, in radians. '0' is forward, clockwise positive when viewed from the top.

Use r2d() to convert.

5.90.3.2 get_mag()

```
double Vector2D::get_mag ( ) const
```

Returns

the magnitude of the vector

Get the magnitude of the vector

5.90.3.3 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.

5.90.3.4 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.

5.90.3.5 normalize()

```
Vector2D Vector2D::normalize ( )
```

Changes the magnitude of the vector to 1

Returns

the normalized vector

Changes the magnetude of the vector to 1

5.90.3.6 operator*()

```
Vector2D Vector2D::operator* (
            const double & x )
```

Scales a Vector2D by a scalar with the * operator

Parameters

x	the value to scale the vector by

Returns

the this Vector2D scaled by x

5.90.3.7 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

other	the vector to add to this

Returns

the sum of the vectors

5.90.3.8 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

other	the vector to subtract from this

Returns

the difference of the vectors

5.90.3.9 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:

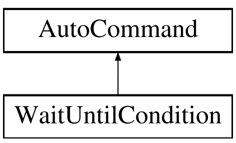
- include/utils/vector2d.h
- src/utils/vector2d.cpp

5.91 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

5.91.1 Detailed Description

Waits until the condition is true.

5.91.2 Member Function Documentation

5.91.2.1 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.

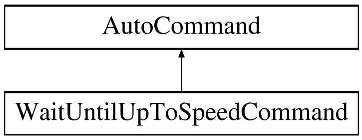
The documentation for this class was generated from the following file:

- include/utils/command_structure/auto_command.h

5.92 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

5.92.1 Detailed Description

AutoCommand that listens to the Flywheel and waits until it is at its target speed +/- the specified threshold

5.92.2 Constructor & Destructor Documentation

5.92.2.1 WaitUntilUpToSpeedCommand()

```
WaitUntilUpToSpeedCommand::WaitUntilUpToSpeedCommand (
            Flywheel & flywheel,
            int threshold_rpm )
```

Creat a WaitUntilUpToSpeedCommand

Parameters

flywheel	the flywheel system we are commanding
threshold_rpm	the threshold over and under the flywheel target RPM that we define to be acceptable

5.92.3 Member Function Documentation

5.92.3.1 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

5.93 screen::WidgetConfig Struct Reference

Public Types

- enum **Type** {
  Col , **Row** , **Slider** , **Button** ,
  Checkbox , **Label** , **Text** , **Graph** }

Public Attributes

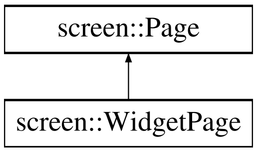
- Type **type**
- union {
  std::vector< SizedWidget > **widgets**
  SliderConfig **slider**
  ButtonConfig **button**
  CheckboxConfig **checkbox**
  LabelConfig **label**
  TextConfig **text**
  GraphDrawer * **graph**
  } **config**

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

- include/subsystems/screen.h

5.94 screen::WidgetPage Class Reference

Inheritance diagram for screen::WidgetPage:



Public Member Functions

- **WidgetPage** (WidgetConfig &cfg)
- void update (bool was_pressed, int x, int y) override

  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) override

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

5.94.1 Member Function Documentation

5.94.1.1 draw()

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

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

Parameters

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

Reimplemented from screen::Page.

5.94.1.2 update()

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

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

Reimplemented from screen::Page.

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

- include/subsystems/screen.h

Chapter 6

File Documentation

6.1 robot_specs.h

```
#pragma once
#include "../core/include/utils/controls/pid.h"
#include "../core/include/utils/controls/feedback_base.h"

typedef struct
{
  double robot_radius;

  double odom_wheel_diam;
  double odom_gear_ratio;
  double dist_between_wheels;

  double drive_correction_cutoff;

  Feedback *drive_feedback;
  Feedback *turn_feedback;
  PID::pid_config_t correction_pid;

} robot_specs_t;
```

6.2 custom_encoder.h

```
#pragma once
#include "vex.h"

class CustomEncoder : public vex::encoder
{
  typedef vex::encoder super;

  public:
  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);


  private:
  double tick_scalar;
};
```

6.3 flywheel.h

```
#pragma once

#include "../core/include/utils/controls/feedforward.h"
#include "vex.h"
#include "../core/include/robot_specs.h"
#include "../core/include/utils/controls/pid.h"
#include "../core/include/utils/command_structure/auto_command.h"
#include "../core/include/subsystems/screen.h"
#include <atomic>

class Flywheel
{

public:
  // CONSTRUCTORS, GETTERS, AND SETTERS
  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()
  {
    return fb.is_on_target();
  }

  screen::Page *Page() const;

  AutoCommand *SpinRpmCmd(int rpm)
  {

    return new FunctionCommand([this, rpm]()
                               {spin_rpm(rpm); return true; });
  }

  AutoCommand *WaitUntilUpToSpeedCmd()
  {
    return new WaitUntilCondition(
        new FunctionCondition([this]()
                              { return is_on_target(); }));
  }

private:
  friend class FlywheelPage;
  friend int spinRPMTask(void *wheelPointer);

  vex::motor_group &motors;
  bool task_running = false;
  Feedback &fb;
  FeedForward &ff;
  vex::mutex fb_mut;
  double ratio;
  std::atomic<double> target_rpm;
  task rpm_task;
  Filter &avger;

  // Functions for internal use only
  void set_target(double value);
  double measure_RPM();

  void spin_raw(double speed, directionType dir = fwd);
};
```

6.4 layout.h

```
#include <cmath>
#include <functional>

struct SliderCfg{
    double &val;
    double min;
    double max;
```

```
};



```

6.5 lift.h

```
#pragma once

#include "vex.h"
#include "../core/include/utils/controls/pid.h"
#include <iostream>
#include <map>
#include <atomic>
#include <vector>

using namespace vex;
using namespace std;

template <typename T>
class Lift
{
  public:

  struct lift_cfg_t
  {
    double up_speed, down_speed;
    double softstop_up, softstop_down;

    PID::pid_config_t lift_pid_cfg;
  };

  Lift(motor_group &lift_motors, lift_cfg_t &lift_cfg, map<T, double> &setpoint_map, limit *homing_switch=NULL)
  : lift_motors(lift_motors), cfg(lift_cfg), lift_pid(cfg.lift_pid_cfg), setpoint_map(setpoint_map), homing_switch(homing_switch)
  {

    is_async = true;
    setpoint = 0;

    // Create a background task that is constantly updating the lift PID, if requested.
    // Set once, and forget.
    task t([](void* ptr){
      Lift &lift = *((Lift*) ptr);

      while(true)
      {
        if(lift.get_async())
          lift.hold();

        vexDelay(50);
      }

      return 0;
    }, this);

  }

  void control_continuous(bool up_ctrl, bool down_ctrl)
  {
    static timer tmr;

    double cur_pos = 0;

    // Check if there’s a hook for a custom sensor. If not, use the motors.
    if(get_sensor == NULL)
      cur_pos = lift_motors.position(rev);
    else
      cur_pos = get_sensor();

    if(up_ctrl && cur_pos < cfg.softstop_up)
    {
      lift_motors.spin(directionType::fwd, cfg.up_speed, volt);
      setpoint = cur_pos + .3;

      // std::cout « "DEBUG OUT: UP " « setpoint « ", " « tmr.time(sec) « ", " « cfg.down_speed « "\n";

      // Disable the PID while going UP.
      is_async = false;
    } else if(down_ctrl && cur_pos > cfg.softstop_down)
    {
```

```
      // Lower the lift slowly, at a rate defined by down_speed
      if(setpoint > cfg.softstop_down)
        setpoint = setpoint - (tmr.time(sec) * cfg.down_speed);
      // std::cout « "DEBUG OUT: DOWN " « setpoint « ", " « tmr.time(sec) « ", " « cfg.down_speed «
"\n";
      is_async = true;
    } else
    {
      // Hold the lift at the last setpoint
      is_async = true;
    }

    tmr.reset();
  }

  void control_manual(bool up_btn, bool down_btn, int volt_up, int volt_down)
  {
    static bool down_hold = false;
    static bool init = true;

    // Allow for setting position while still calling this function
    if(init || up_btn || down_btn)
    {
      init = false;
      is_async = false;
    }

    double rev = lift_motors.position(rotationUnits::rev);

    if(rev < cfg.softstop_down && down_btn)
      down_hold = true;
    else if( !down_btn )
      down_hold = false;

    if(up_btn && rev < cfg.softstop_up)
      lift_motors.spin(directionType::fwd, volt_up, voltageUnits::volt);
    else if(down_btn && rev > cfg.softstop_down && !down_hold)
      lift_motors.spin(directionType::rev, volt_down, voltageUnits::volt);
    else
      lift_motors.spin(directionType::fwd, 0, voltageUnits::volt);

  }

  void control_setpoints(bool up_step, bool down_step, vector<T> pos_list)
  {
    // Make sure inputs are only processed on the rising edge of the button
    static bool up_last = up_step, down_last = down_step;

    bool up_rising = up_step && !up_last;
    bool down_rising = down_step && !down_last;

    up_last = up_step;
    down_last = down_step;

    static int cur_index = 0;

    // Avoid an index overflow. Shouldn't happen unless the user changes pos_list between calls.
    if(cur_index >= pos_list.size())
      cur_index = pos_list.size() - 1;

    // Increment or decrement the index of the list, bringing it up or down.
    if(up_rising && cur_index < (pos_list.size() - 1))
      cur_index++;
    else if(down_rising && cur_index > 0)
      cur_index--;

    // Set the lift to hold the position in the background with the PID loop
    set_position(pos_list[cur_index]);
    is_async = true;

  }

  bool set_position(T pos)
  {
    this->setpoint = setpoint_map[pos];
    is_async = true;

    return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
  }

  bool set_setpoint(double val)
  {
    this->setpoint = val;
    return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
  }

  double get_setpoint()
```

```
  {
    return this->setpoint;
  }

  void hold()
  {
    lift_pid.set_target(setpoint);
    // std::cout « "DEBUG OUT: SETPOINT " « setpoint « "\n";

    if(get_sensor != NULL)
      lift_pid.update(get_sensor());
    else
      lift_pid.update(lift_motors.position(rev));

    // std::cout « "DEBUG OUT: ROTATION " « lift_motors.rotation(rev) « "\n\n";

    lift_motors.spin(fwd, lift_pid.get(), volt);
  }

  void home()
  {
    static timer tmr;
    tmr.reset();

    while(tmr.time(sec) < 3)
    {
      lift_motors.spin(directionType::rev, 6, volt);

      if (homing_switch == NULL && lift_motors.current(currentUnits::amp) > 1.5)
        break;
      else if (homing_switch != NULL && homing_switch->pressing())
        break;
    }

    if(reset_sensor != NULL)
      reset_sensor();

    lift_motors.resetPosition();
    lift_motors.stop();

  }

  bool get_async()
  {
    return is_async;
  }

  void set_async(bool val)
  {
    this->is_async = val;
  }

  void set_sensor_function(double (*fn_ptr) (void))
  {
    this->get_sensor = fn_ptr;
  }

  void set_sensor_reset(void (*fn_ptr) (void))
  {
    this->reset_sensor = fn_ptr;
  }

  private:

  motor_group &lift_motors;
  lift_cfg_t &cfg;
  PID lift_pid;
  map<T, double> &setpoint_map;
  limit *homing_switch;

  atomic<double> setpoint;
  atomic<bool> is_async;

  double (*get_sensor)(void) = NULL;
  void (*reset_sensor)(void) = NULL;


};
```

6.6 mecanum_drive.h

```
#pragma once

```

```
#include "vex.h"
#include "../core/include/utils/controls/pid.h"

#ifndef PI
#define PI 3.141592654
#endif

class MecanumDrive
{

  public:

  struct mecanumdrive_config_t
  {
    // PID configurations for autonomous driving
    PID::pid_config_t drive_pid_conf;
    PID::pid_config_t drive_gyro_pid_conf;
    PID::pid_config_t turn_pid_conf;

    // Diameter of the mecanum wheels
    double drive_wheel_diam;

    // Diameter of the perpendicular undriven encoder wheel
    double lateral_wheel_diam;

    // Width between the center of the left and right wheels
    double wheelbase_width;

  };

  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);

  private:

  vex::motor &left_front, &right_front, &left_rear, &right_rear;

  mecanumdrive_config_t *config;
  vex::rotation *lateral_wheel;
  vex::inertial *imu;

  PID *drive_pid = NULL;
  PID *drive_gyro_pid = NULL;
  PID *turn_pid = NULL;

  bool init = true;

};
```

6.7 odometry_3wheel.h

```
#pragma once
#include "../core/include/subsystems/odometry/odometry_base.h"
#include "../core/include/subsystems/tank_drive.h"
#include "../core/include/subsystems/custom_encoder.h"

class Odometry3Wheel : public OdometryBase
{
    public:

    typedef struct
    {
        double wheelbase_dist;
        double off_axis_center_dist;
        double wheel_diam;
    } odometry3wheel_cfg_t;

    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);
```

```cpp

    private:

    static pose_t calculate_new_pos(double lside_delta_deg, double rside_delta_deg, double offax_delta_deg, pose_t old_pos, odometry3wheel_cfg_t cfg);

    CustomEncoder &lside_fwd, &rside_fwd, &off_axis;
    odometry3wheel_cfg_t &cfg;


};
```

6.8 odometry_base.h

```cpp
#pragma once

#include "vex.h"
#include "../core/include/utils/geometry.h"
#include "../core/include/robot_specs.h"
#include "../core/include/utils/command_structure/auto_command.h"

#ifndef PI
#define PI 3.141592654
#endif



class OdometryBase
{
public:

    OdometryBase(bool is_async);

    pose_t get_position(void);

    virtual void set_position(const pose_t& newpos=zero_pos);
    AutoCommand *SetPositionCmd(const pose_t& newpos=zero_pos);
    virtual pose_t update() = 0;

    static int background_task(void* ptr);

    void end_async();

    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);

    bool end_task = false;

    double get_speed();

    double get_accel();

    double get_angular_speed_deg();

    double get_angular_accel_deg();

    inline static constexpr pose_t zero_pos = {.x=0.0L, .y=0.0L, .rot=90.0L};

protected:
    vex::task *handle;

    vex::mutex mut;

    pose_t current_pos;

    double speed;
    double accel;
    double ang_speed_deg;
    double ang_accel_deg;
};
```

6.9 odometry_tank.h

```cpp
#pragma once

#include "../core/include/subsystems/odometry/odometry_base.h"
```

```
#include "../core/include/subsystems/custom_encoder.h"
#include "../core/include/utils/geometry.h"
#include "../core/include/utils/vector2d.h"
#include "../core/include/utils/moving_average.h"
#include "../core/include/robot_specs.h"
static int background_task(void* odom_obj);
class OdometryTank : public OdometryBase
{
public:
    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;
private:
    static pose_t calculate_new_pos(robot_specs_t &config, pose_t &stored_info, double lside_diff, double rside_diff, double angle_deg);
    vex::motor_group *left_side, *right_side;
    CustomEncoder *left_custom_enc, *right_custom_enc;
    vex::encoder *left_vex_enc, *right_vex_enc;
    vex::inertial *imu;
    robot_specs_t &config;
    double rotation_offset = 0;
    ExponentialMovingAverage ema = ExponentialMovingAverage(3);
};
```

6.10 screen.h

```
#pragma once
#include "vex.h"
#include <vector>
#include <functional>
#include <map>
#include <cassert>
#include "../core/include/subsystems/odometry/odometry_base.h"
#include "../core/include/utils/graph_drawer.h"
#include "../core/include/utils/controls/pid.h"
#include "../core/include/utils/controls/pidff.h"
namespace screen
{
    class ButtonWidget
    {
    public:
        ButtonWidget(std::function<void(void)> onpress, Rect rect, std::string name) : onpress(onpress), rect(rect), name(name) {}
        ButtonWidget(void (*onpress)(), Rect rect, std::string name) : onpress(onpress), rect(rect), name(name) {}
        bool update(bool was_pressed, int x, int y);
        void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
    private:
        std::function<void(void)> onpress;
        Rect rect;
        std::string name = "";
        bool was_pressed_last = false;
    };
    class SliderWidget
    {
    public:
        SliderWidget(double &val, double low, double high, Rect rect, std::string name) : value(val), low(low), high(high), rect(rect), name(name) {}
        bool update(bool was_pressed, int x, int y);
```

```
        void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);

    private:
        double &value;

        double low;
        double high;

        Rect rect;
        std::string name = "";
    };

    struct WidgetConfig;

    struct SliderConfig
    {
        double &val;
        double low;
        double high;
    };
    struct ButtonConfig
    {
        std::function<void()> onclick;
    };
    struct CheckboxConfig
    {
        std::function<void(bool)> onupdate;
    };
    struct LabelConfig
    {
        std::string label;
    };

    struct TextConfig
    {
        std::function<std::string()> text;
    };
    struct SizedWidget
    {
        int size;
        WidgetConfig &widget;
    };
    struct WidgetConfig
    {
        enum Type
        {
            Col,
            Row,
            Slider,
            Button,
            Checkbox,
            Label,
            Text,
            Graph,
        };
        Type type;
        union
        {
            std::vector<SizedWidget> widgets;
            SliderConfig slider;
            ButtonConfig button;
            CheckboxConfig checkbox;
            LabelConfig label;
            TextConfig text;
            GraphDrawer *graph;
        } config;
    };

    class Page;
    class Page
    {
    public:
        virtual void update(bool was_pressed, int x, int y);
        virtual void draw(vex::brain::lcd &screen, bool first_draw,
                          unsigned int frame_number);
    };

    struct ScreenRect
    {
        uint32_t x1;
        uint32_t y1;
        uint32_t x2;
        uint32_t y2;
    };
    void draw_widget(WidgetConfig &widget, ScreenRect rect);

    class WidgetPage : public Page
```

```
{
public:
    WidgetPage(WidgetConfig &cfg) : base_widget(cfg) {}
    void update(bool was_pressed, int x, int y) override;

    void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override
    {
        draw_widget(base_widget, {.x1 = 20, .y1 = 0, .x2 = 440, .y2 = 240});
    }

private:
    WidgetConfig &base_widget;
};

void start_screen(vex::brain::lcd &screen, std::vector<Page *> pages, int first_page = 0);


void next_page();
void prev_page();

void stop_screen();

using update_func_t = std::function<void(bool, int, int)>;

using draw_func_t = std::function<void(vex::brain::lcd &screen, bool, unsigned int)>;

class StatsPage : public Page
{
public:
    StatsPage(std::map<std::string, vex::motor &> motors);
    void update(bool was_pressed, int x, int y) override;
    void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;

private:
    void draw_motor_stats(const std::string &name, vex::motor &mot, unsigned int frame, int x, int
 y, vex::brain::lcd &scr);

    std::map<std::string, vex::motor &> motors;
    static const int y_start = 0;
    static const int per_column = 4;
    static const int row_height = 20;
    static const int row_width = 200;
};

class OdometryPage : public Page
{
public:
    OdometryPage(OdometryBase &odom, double robot_width, double robot_height, bool do_trail);
    void update(bool was_pressed, int x, int y) override;
    void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;

private:
    static const int path_len = 40;
    static constexpr char const *field_filename = "vex_field_240p.png";

    OdometryBase &odom;
    double robot_width;
    double robot_height;
    uint8_t *buf = nullptr;
    int buf_size = 0;
    pose_t path[path_len];
    int path_index = 0;
    bool do_trail;
    GraphDrawer velocity_graph;
};

class FunctionPage : public Page
{
public:
    FunctionPage(update_func_t update_f, draw_func_t draw_t);
    void update(bool was_pressed, int x, int y) override;
    void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;

private:
    update_func_t update_f;
    draw_func_t draw_f;
};

class PIDPage : public Page
{
public:
    PIDPage(
        PID &pid, std::string name, std::function<void(void)> onchange = []() {});
    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;

    private:
        void zero_d_f() { cfg.d = 0; }
        void zero_i_f() { cfg.i = 0; }

        PID::pid_config_t &cfg;
        PID &pid;
        const std::string name;
        std::function<void(void)> onchange;

        SliderWidget p_slider;
        SliderWidget i_slider;
        SliderWidget d_slider;
        ButtonWidget zero_i;
        ButtonWidget zero_d;

        GraphDrawer graph;
    };

}
```

6.11 tank_drive.h

```
#pragma once

#ifndef PI
#define PI 3.141592654
#endif

#include "vex.h"
#include "../core/include/subsystems/odometry/odometry_tank.h"
#include "../core/include/utils/controls/pid.h"
#include "../core/include/utils/controls/feedback_base.h"
#include "../core/include/robot_specs.h"
#include "../core/include/utils/pure_pursuit.h"
#include "../core/include/utils/command_structure/auto_command.h"
#include <vector>

using namespace vex;

class TankDrive
{
public:
  enum class BrakeType
  {
    None,
    ZeroVelocity,
    Smart,
  };
  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, BrakeType bt = BrakeType::None);
  void drive_tank_raw(double left, double right);

  void drive_arcade(double forward_back, double left_right, int power = 1, BrakeType bt = BrakeType::None);
```

```
  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();
  static double modify_inputs(double input, int power = 2);
  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);
private:
  motor_group &left_motors;
  motor_group &right_motors;
  PID correction_pid;
  Feedback *drive_default_feedback = NULL;
  Feedback *turn_default_feedback = NULL;
  OdometryBase *odometry;
  robot_specs_t &config;
  bool func_initialized = false;
  bool is_pure_pursuit = false;
};
```

6.12 auto_chooser.h

```
#pragma once
#include "vex.h"
#include <string>
#include <vector>
#include "../core/include/subsystems/screen.h"
#include "../core/include/utils/geometry.h"
class AutoChooser : public screen::Page
{
public:
  AutoChooser(std::vector<std::string> paths, size_t def = 0);
  void update(bool was_pressed, int x, int y);
  void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
  size_t get_choice();
protected:
  struct entry_t
  {
    Rect rect;
    std::string name;
  };
  static const size_t width = 380;
  static const size_t height = 220;
  size_t choice;
  std::vector<entry_t> list ;
};
```

6.13 auto_command.h

```
#pragma once

#include "vex.h"
#include <functional>
#include <vector>
#include <queue>
#include <atomic>


class Condition
{
public:
  Condition *Or(Condition *b);
  Condition *And(Condition *b);
  virtual bool test() = 0;
};


class AutoCommand
{
public:
  static constexpr double default_timeout = 10.0;
  virtual bool run() { return true; }
  virtual void on_timeout() {}
  AutoCommand *withTimeout(double t_seconds)
  {
    if (this->timeout_seconds < 0)
    {
      // should never be timed out
      return this;
    }
    this->timeout_seconds = t_seconds;
    return this;
  }
  AutoCommand *withCancelCondition(Condition *true_to_end){
    this->true_to_end = true_to_end;
    return this;
  }
  double timeout_seconds = default_timeout;
  Condition *true_to_end = nullptr;
};

class FunctionCommand : public AutoCommand
{
public:
  FunctionCommand(std::function<bool(void)> f) : f(f) {}
  bool run()
  {
    return f();
  }

private:
  std::function<bool(void)> f;
};

// Times tested 3
// Test 1 -> false
// Test 2 -> false
// Test 3 -> true
// Returns false until the Nth time that it is called
// This is pretty much only good for implementing RepeatUntil
class TimesTestedCondition : public Condition
{
public:
  TimesTestedCondition(size_t N) : max(N) {}
  bool test() override
  {
    count++;
    if (count >= max)
    {
      return true;
    }
    return false;
  }

private:
  size_t count = 0;
  size_t max;
};

class FunctionCondition : public Condition
{
public:
  FunctionCondition(
```

```
      std::function<bool()> cond, std::function<void(void)> timeout = []() {}) : cond(cond),
timeout(timeout)
  {
  }
  bool test() override;

private:
  std::function<bool()> cond;
  std::function<void(void)> timeout;
};

class IfTimePassed : public Condition
{
public:
  IfTimePassed(double time_s);
  bool test() override;

private:
  double time_s;
  vex::timer tmr;
};

class WaitUntilCondition : public AutoCommand
{
public:
  WaitUntilCondition(Condition *cond) : cond(cond) {}
  bool run() override
  {
    return cond->test();
  }

private:
  Condition *cond;
};


class InOrder : public AutoCommand
{
public:
  InOrder(const InOrder &other) = default;
  InOrder(std::queue<AutoCommand *> cmds);
  InOrder(std::initializer_list<AutoCommand *> cmds);
  bool run() override;
  void on_timeout() override;

private:
  AutoCommand *current_command = nullptr;
  std::queue<AutoCommand *> cmds;
  vex::timer tmr;
};

class Parallel : public AutoCommand
{
public:
  Parallel(std::initializer_list<AutoCommand *> cmds);
  bool run() override;
  void on_timeout() override;

private:
  std::vector<AutoCommand *> cmds;
  std::vector<vex::task *> runners;
};

class Branch : public AutoCommand
{
public:
  Branch(Condition *cond, AutoCommand *false_choice, AutoCommand *true_choice);
  ~Branch();
  bool run() override;
  void on_timeout() override;

private:
  AutoCommand *false_choice;
  AutoCommand *true_choice;
  Condition *cond;
  bool choice = false;
  bool chosen = false;
  vex::timer tmr;
};

class Async : public AutoCommand
{
public:
  Async(AutoCommand *cmd) : cmd(cmd) {}
  bool run() override;

private:
```

```
  AutoCommand *cmd = nullptr;
};

class RepeatUntil : public AutoCommand
{
public:
  RepeatUntil(InOrder cmds, size_t repeats);
  RepeatUntil(InOrder cmds, Condition *true_to_end);
  bool run() override;
  void on_timeout() override;

private:
  const InOrder cmds;
  InOrder *working_cmds;
  Condition *cond;
};
```

6.14 basic_command.h

```

#pragma once

#include "../core/include/utils/command_structure/auto_command.h"

//Basic Motor Classes-----------------------------------------------

class BasicSpinCommand : public AutoCommand {
    public:

     //Enumurator for the type of power setting in the motor
     enum type {percent,voltage,veocity};

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

     bool run() override;

    private:

     vex::motor &motor;

     type setting;

     vex::directionType dir;

     double power;
};
class BasicStopCommand : public AutoCommand{
    public:

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

     bool run() override;

    private:

     vex::motor &motor;

     vex::brakeType setting;
};

//Basic Solenoid Commands----------------------------------

class BasicSolenoidSet : public AutoCommand{
    public:

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

     bool run() override;

    private:

     vex::pneumatics &solenoid;

     bool setting;
};
```

6.15 command_controller.h

```cpp
#pragma once
#include <vector>
#include <queue>
#include "../core/include/utils/command_structure/auto_command.h"

class CommandController
{
public:
  [[deprecated("Use list constructor instead.")]] CommandController() : command_queue({}) {}

  CommandController(std::initializer_list<AutoCommand *> cmds) : command_queue(cmds) {}
  [[deprecated("Use list constructor instead. If you need to make a decision before adding new commands, use Branch (https://github.com/RIT-VEX-U/Core/wiki/3-%7C-Utilites#commandcontroller)")]] void add(std::vector<AutoCommand *> cmds);
  void add(AutoCommand *cmd, double timeout_seconds = 10.0);

  [[deprecated("Use list constructor instead. If you need to make a decision before adding new commands, use Branch (https://github.com/RIT-VEX-U/Core/wiki/3-%7C-Utilites#commandcontroller)")]] void
  add(std::vector<AutoCommand *> cmds, double timeout_sec);
  void add_delay(int ms);

  void add_cancel_func(std::function<bool(void)> true_if_cancel);

  void run();

  bool last_command_timed_out();

private:
  std::queue<AutoCommand *> command_queue;
  bool command_timed_out = false;
  std::function<bool()> should_cancel = []()
  { return false; };
};
```

6.16 delay_command.h

```cpp

#pragma once

#include "../core/include/utils/command_structure/auto_command.h"

class DelayCommand: public AutoCommand {
  public:
    DelayCommand(int ms): ms(ms) {}

    bool run() override {
      vexDelay(ms);
      return true;
    }

  private:
    // amount of milliseconds to wait
    int ms;
};
```

6.17 drive_commands.h

```cpp

#pragma once

#include "vex.h"
#include "../core/include/utils/geometry.h"
#include "../core/include/utils/command_structure/auto_command.h"
#include "../core/include/subsystems/tank_drive.h"

using namespace vex;


// ==== DRIVING ====

class DriveForwardCommand: public AutoCommand
{
  public:
    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;
```

```
  private:
    // drive system to run the function on
    TankDrive &drive_sys;

    // feedback controller to use
    Feedback &feedback;

    // parameters for drive_forward
    double inches;
    directionType dir;
    double max_speed;
    double end_speed;
};

class TurnDegreesCommand: public AutoCommand
{
  public:
    TurnDegreesCommand(TankDrive &drive_sys, Feedback &feedback, double degrees, double max_speed = 1, double end_speed = 0);

    bool run() override;
    void on_timeout() override;


  private:
    // drive system to run the function on
    TankDrive &drive_sys;

    // feedback controller to use
    Feedback &feedback;

    // parameters for turn_degrees
    double degrees;
    double max_speed;
    double end_speed;
};

class DriveToPointCommand: public AutoCommand
{
  public:
    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;

  private:
    // drive system to run the function on
    TankDrive &drive_sys;

    void on_timeout() override;


    // feedback controller to use
    Feedback &feedback;

    // parameters for drive_to_point
    double x;
    double y;
    directionType dir;
    double max_speed;
    double end_speed;

};

class TurnToHeadingCommand: public AutoCommand
{
  public:
    TurnToHeadingCommand(TankDrive &drive_sys, Feedback &feedback, double heading_deg, double speed = 1, double end_speed = 0);

    bool run() override;
    void on_timeout() override;


  private:
    // drive system to run the function on
    TankDrive &drive_sys;

    // feedback controller to use
    Feedback &feedback;

    // parameters for turn_to_heading
    double heading_deg;
    double max_speed;
```

```
    double end_speed;
};

class PurePursuitCommand: public AutoCommand
{
  public:
  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;

  private:
  TankDrive &drive_sys;
  PurePursuit::Path path;
  directionType dir;
  Feedback &feedback;
  double max_speed;
  double end_speed;

};

class DriveStopCommand: public AutoCommand
{
  public:
    DriveStopCommand(TankDrive &drive_sys);

    bool run() override;
    void on_timeout() override;

  private:
    // drive system to run the function on
    TankDrive &drive_sys;
};


// ==== ODOMETRY ====

class OdomSetPosition: public AutoCommand
{
  public:
    OdomSetPosition(OdometryBase &odom, const pose_t &newpos=OdometryBase::zero_pos);

    bool run() override;

  private:
    // drive system with an odometry config
    OdometryBase &odom;
    pose_t newpos;
};
```

6.18 flywheel_commands.h

```

#pragma once

#include "../core/include/subsystems/flywheel.h"
#include "../core/include/utils/command_structure/auto_command.h"

class SpinRPMCommand: public AutoCommand {
  public:
  SpinRPMCommand(Flywheel &flywheel, int rpm);

    bool run() override;

  private:
    // Flywheel instance to run the function on
    Flywheel &flywheel;

    // parameters for spin_rpm
    int rpm;
};

class WaitUntilUpToSpeedCommand: public AutoCommand {
  public:
    WaitUntilUpToSpeedCommand(Flywheel &flywheel, int threshold_rpm);

    bool run() override;

  private:
    // Flywheel instance to run the function on
    Flywheel &flywheel;
```

```

    // if the actual speed is equal to the desired speed +/- this value, we are ready to fire
    int threshold_rpm;
};

class FlywheelStopCommand: public AutoCommand {
  public:
  FlywheelStopCommand(Flywheel &flywheel);

    bool run() override;

  private:
    // Flywheel instance to run the function on
    Flywheel &flywheel;
};

class FlywheelStopMotorsCommand: public AutoCommand {
  public:
  FlywheelStopMotorsCommand(Flywheel &flywheel);

    bool run() override;

  private:
    // Flywheel instance to run the function on
    Flywheel &flywheel;
};

class FlywheelStopNonTasksCommand: public AutoCommand {
  FlywheelStopNonTasksCommand(Flywheel &flywheel);

    bool run() override;

  private:
    // Flywheel instance to run the function on
    Flywheel &flywheel;
};
```

6.19 bang_bang.h

```
#include "../core/include/utils/controls/feedback_base.h"

class BangBang : public Feedback
{

public:
    BangBang(double thresshold, double low, double high);
    void init(double start_pt, double set_pt, double start_vel [[maybe_unused]] = 0.0, double end_vel
[[maybe_unused]] = 0.0) override;

    double update(double val) override;

    double get() override;

    void set_limits(double lower, double upper) override;

    bool is_on_target() override;

private:
    double setpt;
    double sensor_val;
    double lower_bound, upper_bound;
    double last_output;
    double threshhold;
};
```

6.20 feedback_base.h

```
#pragma once

class Feedback
{
public:
    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;


};
```

6.21 feedforward.h

```
#pragma once

#include <math.h>
#include <vector>
#include "../core/include/utils/math_util.h"
#include "../core/include/utils/moving_average.h"
#include "vex.h"

class FeedForward
{
    public:

    typedef struct
    {
        double kS;
        double kV;
        double kA;
        double kG;
    } ff_config_t;


    FeedForward(ff_config_t &cfg) : cfg(cfg) {}

    double calculate(double v, double a, double pid_ref=0.0)
    {
        double ks_sign = 0;
        if(v != 0)
            ks_sign = sign(v);
        else if(pid_ref != 0)
            ks_sign = sign(pid_ref);

        return (cfg.kS * ks_sign) + (cfg.kV * v) + (cfg.kA * a) + cfg.kG;
    }

    private:

    ff_config_t &cfg;

};


FeedForward::ff_config_t tune_feedforward(vex::motor_group &motor, double pct, double duration);
```

6.22 motion_controller.h

```
#pragma once
#include "../core/include/utils/controls/pid.h"
#include "../core/include/utils/controls/feedforward.h"
#include "../core/include/utils/controls/trapezoid_profile.h"
#include "../core/include/utils/controls/feedback_base.h"
#include "../core/include/subsystems/tank_drive.h"
#include "../core/include/subsystems/screen.h"

#include "vex.h"

class MotionController : public Feedback
{
    public:

    typedef struct
    {
        double max_v;
        double accel;
        PID::pid_config_t pid_cfg;
        FeedForward::ff_config_t ff_cfg;
    } m_profile_cfg_t;

    MotionController(m_profile_cfg_t &config);

    void init(double start_pt, double end_pt, double start_vel, double end_vel) override;
```

```
  double update(double sensor_val) override;

  double get() override;

  void set_limits(double lower, double upper) override;

  bool is_on_target() override;

  motion_t get_motion() const;


  screen::Page *Page();

  static FeedForward::ff_config_t tune_feedforward(TankDrive &drive, OdometryTank &odometry, double
pct=0.6, double duration=2);

  private:

  m_profile_cfg_t config;

  PID pid;
  FeedForward ff;
  TrapezoidProfile profile;

  double current_pos;
  double end_pt;

  double lower_limit = 0, upper_limit = 0;
  double out = 0;
  motion_t cur_motion;

  vex::timer tmr;
  friend class MotionControllerPage;

};
```

6.23 pid.h

```
#pragma once

#include "../core/include/utils/controls/feedback_base.h"
#include "vex.h"
#include <cmath>

using namespace vex;

class PID : public Feedback {
public:
  enum ERROR_TYPE {
    LINEAR,
    ANGULAR // assumes degrees
  };
  struct pid_config_t {
    double p;
    double i;
    double d;
    double deadband;
    double on_target_time;
    ERROR_TYPE error_method;
  };

  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() const;

  double get() override;

  void set_limits(double lower, double upper) override;

  bool is_on_target() override;

  void reset();

  double get_error();

  double get_target() const;

```

```
  void set_target(double target);

  pid_config_t
      &config;

private:
  double last_error =
      0;
  double accum_error =
      0;

  double last_time = 0;
  double on_target_last_time =
      0;

  double lower_limit =
      0;
  double upper_limit =
      0;

  double target = 0;
  double target_vel = 0;
  double sensor_val = 0;
  double out = 0;

  bool is_checking_on_target =
      false;

  timer pid_timer;
};
```

6.24 pidff.h

```
#pragma once
#include "../core/include/utils/controls/feedback_base.h"
#include "../core/include/utils/controls/feedforward.h"
#include "../core/include/utils/controls/pid.h"

class PIDFF : public Feedback {
public:
  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 get_target() const;
  double get_sensor_val() const;
  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;

  void reset();

  PID pid;

private:
  FeedForward::ff_config_t &ff_cfg;

  FeedForward ff;

  double out;
  double lower_lim, upper_lim;
};
```

6.25 take_back_half.h

```
#pragma once
#include "../core/include/utils/controls/feedback_base.h"

class TakeBackHalf : public Feedback
{
```

```
public:
    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;

    double TBH_gain;
    double first_cross_split;
private:
    double on_target_threshhold;

    double target = 0.0;

    bool first_cross = true;
    double tbh = 0.0;
    double prev_error = 0.0;

    double output = 0.0;
    double lower = 0.0, upper = 0.0;
};
```

6.26 trapezoid_profile.h

```
#pragma once

const int MAX_TRAPEZOID_PROFILE_SEGMENTS = 4;

typedef struct {
  double pos;
  double vel;
  double accel;

} motion_t;

typedef struct {
  double pos_after;
  double vel_after;
  double accel;
  double duration;
} trapezoid_profile_segment_t;

class TrapezoidProfile {
public:
  TrapezoidProfile(double max_v, double accel);

  motion_t calculate(double time_s, double pos_s);

  motion_t calculate_time_based(double time_s);

  void set_endpts(double start, double end);

  void set_vel_endpts(double start, double end);

  void set_accel(double accel);

  void set_max_v(double max_v);

  double get_movement_time() const;

  double get_max_v() const;
  double get_accel() const;

private:
  double si, sf;
  double vi, vf;
  double max_v;
  double accel;
  double duration;

  trapezoid_profile_segment_t segments[MAX_TRAPEZOID_PROFILE_SEGMENTS];
  int num_acceleration_phases;

  bool precalculated;

  bool precalculate();

  trapezoid_profile_segment_t calculate_kinetic_motion(double si, double vi,
```

```
                                                    double v_target);

  trapezoid_profile_segment_t calculate_next_segment(double s, double v);
};
```

6.27 generic_auto.h

```
#pragma once

#include <queue>
#include <map>
#include "vex.h"
#include <functional>

typedef std::function<bool(void)> state_ptr;

class GenericAuto
{
  public:

  [[deprecated("Use CommandController instead.")]]
  bool run(bool blocking);

  [[deprecated("Use CommandController instead.")]]
  void add(state_ptr new_state);

  [[deprecated("Use CommandController instead.")]]
  void add_async(state_ptr async_state);

  [[deprecated("Use CommandController instead.")]]
  void add_delay(int ms);

  private:

  std::queue<state_ptr> state_list;

};
```

6.28 geometry.h

```
#pragma once
#include <cmath>

struct point_t
{
    double x;
    double y;

    double dist(const point_t other) const
    {
        return std::sqrt(std::pow(this->x - other.x, 2) + pow(this->y - other.y, 2));
    }

    point_t operator+(const point_t &other) const
    {
        point_t p{
            .x = this->x + other.x,
            .y = this->y + other.y};
        return p;
    }

    point_t operator-(const point_t &other) const
    {
        point_t p{
            .x = this->x - other.x,
            .y = this->y - other.y};
        return p;
    }

    point_t operator*(double s) const
    {
        return {x * s, y * s};
    }
    point_t operator/(double s) const
    {
        return {x / s, y / s};
    }

    point_t operator-() const
```

```
    {
        return {-x, -y};
    }
    point_t operator+() const
    {
        return {x, y};
    }

    bool operator==(const point_t &rhs)
    {
        return x == rhs.x && y == rhs.y;
    }
};

struct pose_t
{
    double x;
    double y;
    double rot;

    point_t get_point()
    {
        return point_t{.x = x, .y = y};
    }

} ;

struct Rect
{
    point_t min;
    point_t max;
    static Rect from_min_and_size(point_t min, point_t size){
        return {min, min+size};
    }
    point_t dimensions() const
    {
        return max - min;
    }
    point_t center() const{
        return (min + max)/2;
    }
    double width() const{
        return max.x - min.x;
    }
    double height() const{
        return max.y - min.y;
    }
    bool contains(point_t p) const
    {
        bool xin = p.x > min.x && p.x < max.x;
        bool yin = p.y > min.y && p.y < max.y;
        return xin && yin;
    }

};

struct Mat2
{
    double X11, X12;
    double X21, X22;
    point_t operator*(const point_t p) const
    {
        double outx = p.x * X11 + p.y * X12;
        double outy = p.x * X21 + p.y * X22;
        return {outx, outy};
    }

    static Mat2 FromRotationDegrees(double degrees)
    {
        double rad = degrees * (M_PI / 180.0);
        double c = cos(rad);
        double s = sin(rad);
        return {c, -s, s, c};
    }
};
```

6.29 graph_drawer.h

```
#pragma once

#include <string>
#include <stdio.h>
#include <vector>
```

```
#include <cmath>
#include "vex.h"
#include "../core/include/utils/geometry.h"
#include "../core/include/utils/vector2d.h"

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

private:
  std::vector<std::vector<point_t» series;
  int sample_index = 0;
  std::vector<vex::color> cols;
  vex::color bgcol = vex::transparent;
  bool border;
  double upper;
  double lower;
  bool auto_fit = false;
};
```

6.30 logger.h

```
#pragma once

#include <cstdarg>
#include <cstdio>
#include <string>
#include "vex.h"

enum LogLevel
{
    DEBUG,
    NOTICE,
    WARNING,
    ERROR,
    CRITICAL,
    TIME
};

class Logger
{
private:
    const std::string filename;
    vex::brain::sdcard sd;
    void write_level(LogLevel l);

public:
    static constexpr int MAX_FORMAT_LEN = 512;
    explicit Logger(const std::string &filename);

    Logger(const Logger &l) = delete;
    Logger &operator=(const Logger &l) = delete;


    void Log(const std::string &s);

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

    void Logln(const std::string &s);

    void Logln(LogLevel level, const std::string &s);

    void Logf(const char *fmt, ...);

    void Logf(LogLevel level, const char *fmt, ...);
};
```

6.31 math_util.h

```
#pragma once
#include <vector>
#include "math.h"
```

```
#include "vex.h"
#include "../core/include/utils/geometry.h"


double clamp(double value, double low, double high);

double lerp(double a, double b, double t);
double sign(double x);

double wrap_angle_deg(double input);
double wrap_angle_rad(double input);

/*
Calculates the variance of  a set of numbers (needed for linear regression)
https://en.wikipedia.org/wiki/Variance
@param values   the values for which the variance is taken
@param mean     the average of values
*/
double variance(std::vector<double> const &values, double mean);


/*
Calculates the average of a vector of doubles
@param values   the list of values for which the average is taken
*/
double mean(std::vector<double> const &values);

/*
Calculates the covariance of a set of points (needed for linear regression)
https://en.wikipedia.org/wiki/Covariance

@param points   the points for which the covariance is taken
@param meanx    the mean value of all x coordinates in points
@param meany    the mean value of all y coordinates in points
*/
double covariance(std::vector<std::pair<double, double» const &points, double meanx, double meany);

/*
Calculates the slope and y intercept of the line of best fit for the data
@param points the points for the data
*/
std::pair<double, double> calculate_linear_regression(std::vector<std::pair<double, double» const &points);

double estimate_path_length(const std::vector<point_t> &points);
```

6.32 moving_average.h

```
#pragma once
#include <vector>

class Filter
{
public:
  virtual void add_entry(double n) = 0;
  virtual double get_value() const = 0;
};

class MovingAverage : public Filter
{
public:
  /*
   * Create a moving average calculator with 0 as the default value
   *
   * @param buffer_size    The size of the buffer. The number of samples that constitute a valid reading
   */
  MovingAverage(int buffer_size);
  /*
   * Create a moving average calculator with a specified default value
   * @param buffer_size    The size of the buffer. The number of samples that constitute a valid reading
   * @param starting_value The value that the average will be before any data is added
   */
  MovingAverage(int buffer_size, double starting_value);

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

```
     *      ^
     * @param n  the sample that will be added to the moving average.
     */
    void add_entry(double n) override;

    double get_value() const override;

    int get_size() const;

  private:
    int buffer_index;            // index of the next value to be overridden
    std::vector<double> buffer; // all current data readings we've taken
    double current_avg;          // the current value of the data
  };

  class ExponentialMovingAverage : public Filter
  {
  public:
    /*
     * Create a moving average calculator with 0 as the default value
     *
     * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
  reading
     */
    ExponentialMovingAverage(int buffer_size);
    /*
     * Create a moving average calculator with a specified default value
     * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
  reading
     * @param starting_value The value that the average will be before any data is added
     */
    ExponentialMovingAverage(int buffer_size, double starting_value);

    /*
     * Add a reading to the buffer
     * Before:
     * [ 1 1 2 2 3 3] => 2
     *    ^
     * After:
     * [ 2 1 2 2 3 3] => 2.16
     *      ^
     * @param n  the sample that will be added to the moving average.
     */
    void add_entry(double n) override;

    double get_value() const override;

    int get_size();

  private:
    int buffer_index;            // index of the next value to be overridden
    std::vector<double> buffer; // all current data readings we've taken
    double current_avg;          // the current value of the data
  };
```

6.33 pure_pursuit.h

```
#pragma once

#include <vector>
#include "../core/include/utils/geometry.h"
#include "../core/include/utils/vector2d.h"
#include "vex.h"

using namespace vex;

namespace PurePursuit {
  class Path
  {
    public:
      Path(std::vector<point_t> points, double radius);

      std::vector<point_t> get_points();

      double get_radius();

      bool is_valid();

    private:
      std::vector<point_t> points;
      double radius;
      bool valid;
  };
```

```
  struct spline
  {
    double a, b, c, d, x_start, x_end;

    double getY(double x) {
      return a * pow((x - x_start), 3) + b * pow((x - x_start), 2) + c * (x - x_start) + d;
    }
  };
  struct hermite_point
  {
    double x;
    double y;
    double dir;
    double mag;

    point_t getPoint() const {
      return {x, y};
    }

    Vector2D getTangent() const {
      return Vector2D(dir, mag);
    }
  };

  extern std::vector<point_t> line_circle_intersections(point_t center, double r, point_t point1, point_t point2);
  extern point_t get_lookahead(const std::vector<point_t> &path, pose_t robot_loc, double radius);

  extern std::vector<point_t> inject_path(const std::vector<point_t> &path, double spacing);

  extern std::vector<point_t> smooth_path(const std::vector<point_t> &path, double weight_data, double weight_smooth, double tolerance);

  extern std::vector<point_t> smooth_path_cubic(const std::vector<point_t> &path, double res);

  extern std::vector<point_t> smooth_path_hermite(const std::vector<hermite_point> &path, double step);

  extern double estimate_remaining_dist(const std::vector<point_t> &path, pose_t robot_pose, double radius);

}
```

6.34 serializer.h

```
#pragma once
#include <algorithm>
#include <map>
#include <string>
#include <vector>
#include <stdio.h>
#include <vex.h>

const char serialization_separator = '$';
const std::size_t MAX_FILE_SIZE = 4096;

class Serializer
{
private:
    bool flush_always;
    std::string filename;
    std::map<std::string, int> ints;
    std::map<std::string, bool> bools;
    std::map<std::string, double> doubles;
    std::map<std::string, std::string> strings;

    bool read_from_disk();

public:
    ~Serializer()
    {
        save_to_disk();
        printf("Saving %s\n", filename.c_str());
        fflush(stdout);
    }

    explicit Serializer(const std::string &filename, bool flush_always = true) : flush_always(flush_always), filename(filename), ints({}), bools({}), doubles({}), strings({})

    {
        read_from_disk();
    }

```

```
    void save_to_disk() const;


    void set_int(const std::string &name, int i);

    void set_bool(const std::string &name, bool b);

    void set_double(const std::string &name, double d);

    void set_string(const std::string &name, std::string str);


    int int_or(const std::string &name, int otherwise);

    bool bool_or(const std::string &name, bool otherwise);

    double double_or(const std::string &name, double otherwise);

    std::string string_or(const std::string &name, std::string otherwise);
};
```

6.35 vector2d.h

```
#pragma once


#include <cmath>
#include "../core/include/utils/geometry.h"

#ifndef PI
#define PI 3.141592654
#endif
class Vector2D
{
public:
    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);

private:

    double dir, mag;

};

double deg2rad(double deg);

double rad2deg(double r);
```

Index