# 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

# Namespace Index

## 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

# Chapter 3

# Hierarchical Index

## 3.1 Class Hierarchy

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

# Chapter 4

# Class Index

## 4.1 Class List

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

# Chapter 5

# File Index

## 5.1 File List

Here is a list of all files with brief descriptions:

# Chapter 6

# Namespace Documentation

## 6.1 PurePursuit Namespace Reference

### Classes

- struct hermite_point
- class Path
- struct spline

### Functions

- std::vector< point_t > line_circle_intersections (point_t center, double r, point_t point1, point_t point2)
- point_t get_lookahead (const std::vector< point_t > &path, pose_t robot_loc, double radius)
- std::vector< point_t > inject_path (const std::vector< point_t > &path, double spacing)
- std::vector< point_t > smooth_path (const std::vector< point_t > &path, double weight_data, double weight_smooth, double tolerance)
- std::vector< point_t > smooth_path_cubic (const std::vector< point_t > &path, double res)
- std::vector< point_t > smooth_path_hermite (const std::vector< hermite_point > &path, double step)
- double estimate_remaining_dist (const std::vector< point_t > &path, pose_t robot_pose, double radius)

### 6.1.1 Function Documentation

#### 6.1.1.1 estimate_remaining_dist()

```
double PurePursuit::estimate_remaining_dist (
            const std::vector< point_t > & path,
            pose_t robot_pose,
            double radius )  [extern]
```

Estimates the remaining distance from the robot's position to the end, by "searching" for the robot along the path and running a "connect the dots" distance algoritm

**Parameters**

| | |
|---|---|
| *path* | The pure pursuit path the robot is following |
| *robot_pose* | The robot's current position |
| *radius* | Pure pursuit "radius", used to search for the robot along the path |

**Returns**

A rough estimate of the remaining distance

#### 6.1.1.2 get_lookahead()

```
point_t PurePursuit::get_lookahead (
            const std::vector< point_t > & path,
            pose_t robot_loc,
            double radius ) [extern]
```

Selects a look ahead from all the intersections in the path.

#### 6.1.1.3 inject_path()

```
std::vector< point_t > PurePursuit::inject_path (
            const std::vector< point_t > & path,
            double spacing ) [extern]
```

Injects points in a path without changing the curvature with a certain spacing.

#### 6.1.1.4 line_circle_intersections()

```
std::vector< point_t > PurePursuit::line_circle_intersections (
            point_t center,
            double r,
            point_t point1,
            point_t point2 ) [extern]
```

Returns points of the intersections of a line segment and a circle. The line segment is defined by two points, and the circle is defined by a center and radius.

#### 6.1.1.5 smooth_path()

```
std::vector< point_t > PurePursuit::smooth_path (
            const std::vector< point_t > & path,
            double weight_data,
            double weight_smooth,
            double tolerance ) [extern]
```

Returns a smoothed path maintaining the start and end of the path.

Weight data is how much weight to update the data (alpha) Weight smooth is how much weight to smooth the coordinates (beta) Tolerance is how much change per iteration is necessary to continue iterating.

Honestly have no idea if/how this works. https://medium.com/@jaems33/understanding-robot-motion-path-

#### 6.1.1.6 smooth_path_cubic()

```
std::vector< point_t > PurePursuit::smooth_path_cubic (
            const std::vector< point_t > & path,
            double res ) [extern]
```

#### 6.1.1.7 smooth_path_hermite()

```
std::vector< point_t > PurePursuit::smooth_path_hermite (
            const std::vector< hermite_point > & path,
            double steps ) [extern]
```

Interpolates a smooth path given a list of waypoints using hermite splines. For more information: https://www.youtube.com/watch?v=hG0p4XgePSA.

**Parameters**

| | |
|---|---|
| *path* | The path of hermite points to interpolate. |
| *steps* | The number of points interpolated between points. |

**Returns**

The smoothed path.

## 6.2 screen Namespace Reference

### Classes

- struct ButtonConfig
- class ButtonWidget

  *Widget that does something when you tap it. The function is only called once when you first tap it.*
- struct CheckboxConfig
- class FunctionPage

  *Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything.*
- struct LabelConfig
- class OdometryPage

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

  *Page describes one part of the screen slideshow.*
- class PIDPage

  *PIDPage provides a way to tune a pid controller on the screen.*
- struct ScreenData

  *The ScreenData class holds the data that will be passed to the screen thread you probably shouldnt have to use it.*
- struct ScreenRect
- struct SizedWidget
- struct SliderConfig
- class SliderWidget

*Widget that updates a double value. Updates by reference so watch out for race conditions cuz the screen stuff lives on another thread.*
- class StatsPage

  *Draws motor stats and battery stats to the screen.*
- struct TextConfig
- struct WidgetConfig
- class WidgetPage

### Typedefs

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

  *type of function needed for update*
- using draw_func_t = std::function<void(vex::brain::lcd &screen, bool, unsigned int)>

  *type of function needed for draw*

### Functions

- void draw_widget (WidgetConfig &widget, ScreenRect rect)
- void start_screen (vex::brain::lcd &screen, std::vector< Page ∗ > pages, int first_page=0)

  *Start the screen background task. Once you start this, no need to draw to the screen manually elsewhere.*
- void next_page ()
- void prev_page ()
- void stop_screen ()

  *stops the screen. If you have a drive team that hates fun call this at the start of opcontrol*
- void draw_label (vex::brain::lcd &scr, std::string lbl, ScreenRect rect)
- void draw_widget (vex::brain::lcd &scr, WidgetConfig &widget, ScreenRect rect)
- int in_to_px (double in)

## 6.2.1 Typedef Documentation

### 6.2.1.1 draw_func_t

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

type of function needed for draw

### 6.2.1.2 update_func_t

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

type of function needed for update

## 6.2.2 Function Documentation

### 6.2.2.1 draw_label()

```
void screen::draw_label (
            vex::brain::lcd & scr,
            std::string lbl,
            ScreenRect rect )
```

#### 6.2.2.2 draw_widget() [1/2]

```
void screen::draw_widget (
            vex::brain::lcd & scr,
            WidgetConfig & widget,
            ScreenRect rect )
```

#### 6.2.2.3 draw_widget() [2/2]

```
void screen::draw_widget (
            WidgetConfig & widget,
            ScreenRect rect )
```

#### 6.2.2.4 in_to_px()

```
int screen::in_to_px (
            double in )
```

#### 6.2.2.5 next_page()

```
void screen::next_page ( )
```

#### 6.2.2.6 prev_page()

```
void screen::prev_page ( )
```

#### 6.2.2.7 start_screen()

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

Start the screen background task. Once you start this, no need to draw to the screen manually elsewhere.

start_screen begins a screen. only call this once per program (a good place is vexcodeInit) This is a set and forget type function. You don't have to wait on it or start it in a new thread

**Parameters**

| | |
|---|---|
| *screen* | reference to the vex screen |
| *pages* | drawing pages |
| *first_page* | optional, which page to start the program at. by default 0 |
| *screen* | the brain screen |
| *pages* | the list of pages in your UI slideshow |
| *first_page* | the page to start on (by default 0) |

#### 6.2.2.8 stop_screen()

```
void screen::stop_screen ( )
```

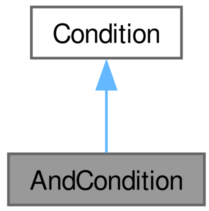
stops the screen. If you have a drive team that hates fun call this at the start of opcontrol

# Chapter 7

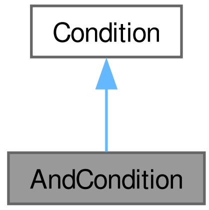
# Class Documentation

## 7.1 AndCondition Class Reference

Inheritance diagram for AndCondition:




Collaboration 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)

## 7.1.1 Constructor & Destructor Documentation

### 7.1.1.1 AndCondition()

```
AndCondition::AndCondition (
            Condition * A,
            Condition * B )  [inline]
```

## 7.1.2 Member Function Documentation

### 7.1.2.1 test()

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

Implements Condition.

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

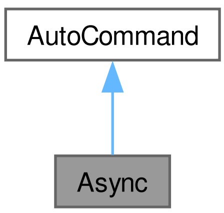
- src/utils/command_structure/auto_command.cpp

# 7.2 Async Class Reference

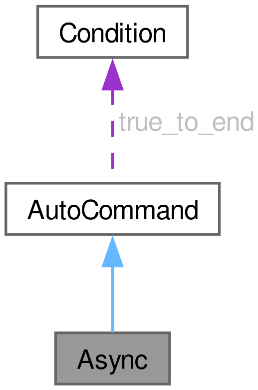
Async runs a command asynchronously will simply let it go and never look back THIS HAS A VERY NICHE USE CASE. THINK ABOUT IF YOU REALLY NEED IT.

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

Inheritance diagram for Async:



Collaboration 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

## 7.2.1 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.

## 7.2.2 Constructor & Destructor Documentation

### 7.2.2.1 Async()

```
Async::Async (
            AutoCommand * cmd )  [inline]
```

## 7.2.3 Member Function Documentation

### 7.2.3.1 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.

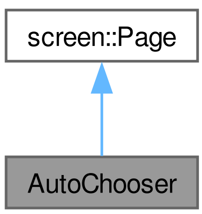
The documentation for this class was generated from the following files:

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

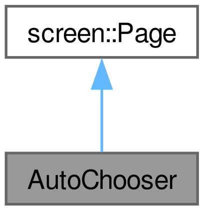
# 7.3 AutoChooser Class Reference

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

Inheritance diagram for AutoChooser:



Collaboration 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

## 7.3.1 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.

## 7.3.2 Constructor & Destructor Documentation

### 7.3.2.1 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 |

## 7.3.3 Member Function Documentation

### 7.3.3.1 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.

### 7.3.3.2 get_choice()

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

Get the currently selected auto choice

**Returns**

the identifier to the auto path

Return the selected autonomous

#### 7.3.3.3 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.

## 7.3.4 Member Data Documentation

### 7.3.4.1 choice

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

the current choice of auto

### 7.3.4.2 height

```
const size_t AutoChooser::height = 220  [static], [protected]
```

### 7.3.4.3 list

```
std::vector<entry_t> AutoChooser::list  [protected]
```

$<$ a list of all possible auto choices

### 7.3.4.4 width

```
const size_t AutoChooser::width = 380  [static], [protected]
```

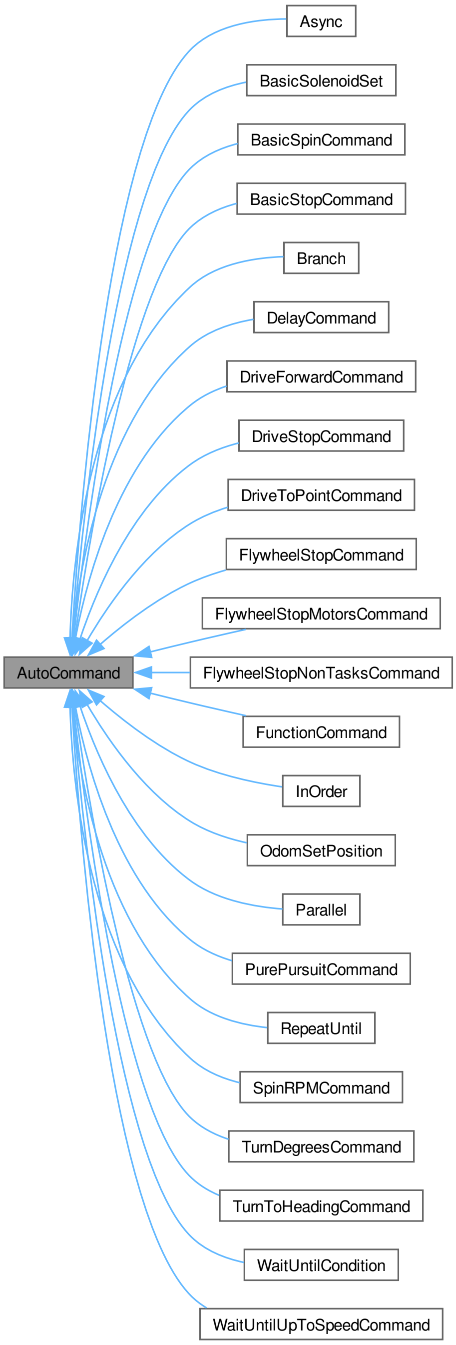
The documentation for this class was generated from the following files:

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

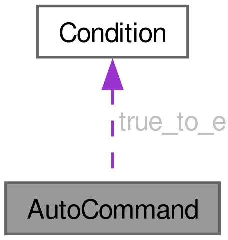
## 7.4 AutoCommand Class Reference

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

Inheritance diagram for AutoCommand:



Collaboration 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

## 7.4.1 Member Function Documentation

### 7.4.1.1 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.

#### 7.4.1.2 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.

#### 7.4.1.3 withCancelCondition()

```
AutoCommand * AutoCommand::withCancelCondition (
            Condition * true_to_end ) [inline]
```

#### 7.4.1.4 withTimeout()

```
AutoCommand * AutoCommand::withTimeout (
            double t_seconds ) [inline]
```

### 7.4.2 Member Data Documentation

#### 7.4.2.1 default_timeout

```
constexpr double AutoCommand::default_timeout = 10.0 [static], [constexpr]
```

#### 7.4.2.2 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...

#### 7.4.2.3 true_to_end

```
Condition* AutoCommand::true_to_end = nullptr
```

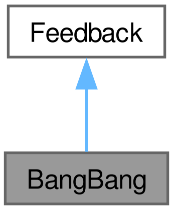
The documentation for this class was generated from the following file:

- include/utils/command_structure/auto_command.h

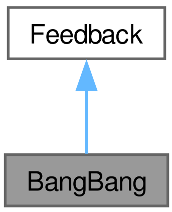
## 7.5 BangBang Class Reference

```
#include <bang_bang.h>
```

Inheritance diagram for BangBang:




Collaboration 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

## 7.5.1 Constructor & Destructor Documentation

### 7.5.1.1 BangBang()

```
BangBang::BangBang (
            double thresshold,
            double low,
            double high )
```

## 7.5.2 Member Function Documentation

### 7.5.2.1 get()

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

**Returns**

the last saved result from the feedback controller

Implements Feedback.

### 7.5.2.2 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.

### 7.5.2.3 is_on_target()

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

**Returns**

true if the feedback controller has reached it's setpoint

Implements Feedback.

#### 7.5.2.4 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.

#### 7.5.2.5 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.

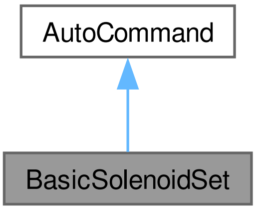
The documentation for this class was generated from the following files:

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

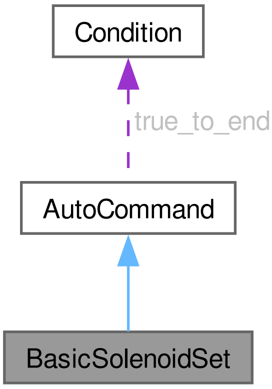
## 7.6 BasicSolenoidSet Class Reference

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

Inheritance diagram for BasicSolenoidSet:




Collaboration 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

## 7.6.1 Detailed Description

AutoCommand wrapper class for BasicSolenoidSet Using the Vex hardware functions

## 7.6.2 Constructor & Destructor Documentation

### 7.6.2.1 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) |

## 7.6.3 Member Function Documentation

### 7.6.3.1 run()

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

Runs the BasicSolenoidSet Overrides run command from AutoCommand.

**Returns**

True Command runs once

Reimplemented from AutoCommand.

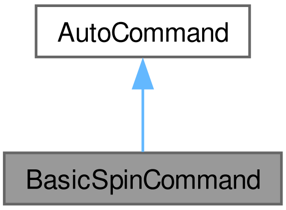
The documentation for this class was generated from the following files:

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

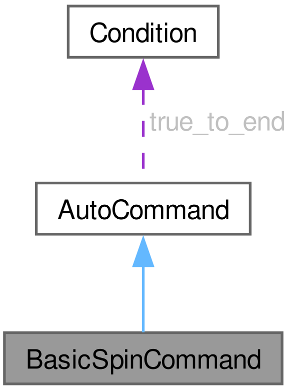
## 7.7 BasicSpinCommand Class Reference

`#include <basic_command.h>`

Inheritance diagram for BasicSpinCommand:




Collaboration 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

## 7.7.1 Detailed Description

AutoCommand wrapper class for BasicSpinCommand using the vex hardware functions

## 7.7.2 Member Enumeration Documentation

### 7.7.2.1 type

```
enum BasicSpinCommand::type
```

**Enumerator**

| | |
|---|---|
| percent | |
| voltage | |
| veocity | |

## 7.7.3 Constructor & Destructor Documentation

### 7.7.3.1 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 |

### 7.7.4 Member Function Documentation

#### 7.7.4.1 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.

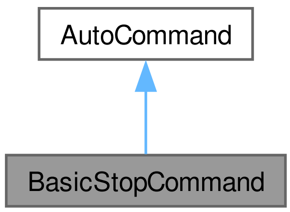
The documentation for this class was generated from the following files:

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

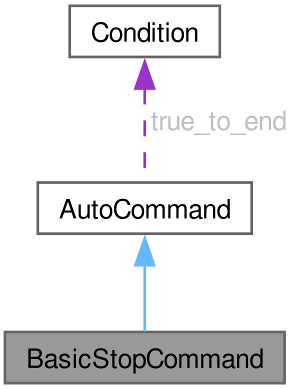
## 7.8 BasicStopCommand Class Reference

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

Inheritance diagram for BasicStopCommand:



Collaboration 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

## 7.8.1 Detailed Description

AutoCommand wrapper class for BasicStopCommand Using the Vex hardware functions

## 7.8.2 Constructor & Destructor Documentation

### 7.8.2.1 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 |

## 7.8.3 Member Function Documentation

### 7.8.3.1 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:

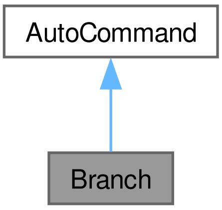
- include/utils/command_structure/basic_command.h
- src/utils/command_structure/basic_command.cpp

## 7.9 Branch Class Reference

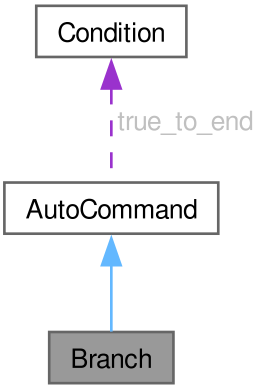
Branch chooses from multiple options at runtime. the function decider returns an index into the choices vector If you wish to make no choice and skip this section, return NO_CHOICE; any choice that is out of bounds set to NO_CHOICE.

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

Inheritance diagram for Branch:




Collaboration diagram for Branch:




### Public Member Functions

- Branch (Condition ∗cond, AutoCommand ∗false_choice, AutoCommand ∗true_choice)
- ∼Branch ()
- 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

## 7.9.1 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.

## 7.9.2 Constructor & Destructor Documentation

### 7.9.2.1 Branch()

```
Branch::Branch (
            Condition * cond,
            AutoCommand * false_choice,
            AutoCommand * true_choice )
```

### 7.9.2.2 ~Branch()

```
Branch::~Branch ( )
```

## 7.9.3 Member Function Documentation

### 7.9.3.1 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.

#### 7.9.3.2 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

## 7.10 screen::ButtonConfig Struct Reference

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

**Public Attributes**

- std::function< void()> onclick

### 7.10.1 Member Data Documentation

#### 7.10.1.1 onclick

```
std::function<void()> screen::ButtonConfig::onclick
```

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

- include/subsystems/screen.h

## 7.11 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*

## 7.11.1 Detailed Description

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

## 7.11.2 Constructor & Destructor Documentation

### 7.11.2.1 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 |

### 7.11.2.2 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 |

### 7.11.3 Member Function Documentation

#### 7.11.3.1 draw()

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

draws the button to the screen

#### 7.11.3.2 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

## 7.12 screen::CheckboxConfig Struct Reference

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

**Public Attributes**

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

### 7.12.1 Member Data Documentation

#### 7.12.1.1 onupdate

`std::function<void(bool)> screen::CheckboxConfig::onupdate`

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

- include/subsystems/screen.h

## 7.13 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 ()

### 7.13.1 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.

### 7.13.2 Constructor & Destructor Documentation

#### 7.13.2.1 CommandController() [1/2]

`CommandController::CommandController ( ) [inline]`

Create an empty CommandController. Add Command with CommandController::add()

#### 7.13.2.2 CommandController() [2/2]

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

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

**Parameters**

| | |
|---|---|
| *cmds* | |

### 7.13.3 Member Function Documentation

#### 7.13.3.1 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 |

#### 7.13.3.2 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 |

#### 7.13.3.3 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 |

#### 7.13.3.4 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 |

#### 7.13.3.5 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 |

#### 7.13.3.6 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

#### 7.13.3.7 run()

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

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

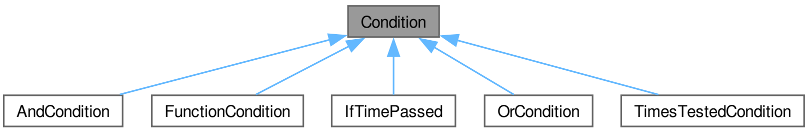
The documentation for this class was generated from the following files:

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

## 7.14 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

### 7.14.1 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

### 7.14.2 Member Function Documentation

#### 7.14.2.1 And()

```
Condition * Condition::And (
            Condition * b )
```

#### 7.14.2.2 Or()

```
Condition * Condition::Or (
            Condition * b )
```

#### 7.14.2.3 test()

```
virtual bool Condition::test ( )  [pure virtual]
```

Implemented in TimesTestedCondition, FunctionCondition, IfTimePassed, OrCondition, and AndCondition.

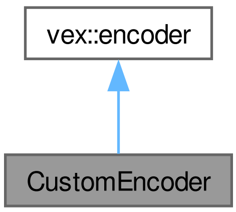
The documentation for this class was generated from the following files:

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

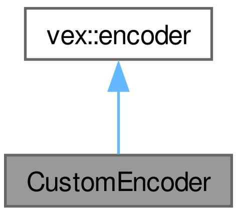
## 7.15 CustomEncoder Class Reference

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

Inheritance diagram for CustomEncoder:




Collaboration 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)

### 7.15.1 Detailed Description

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

### 7.15.2 Constructor & Destructor Documentation

#### 7.15.2.1 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 |

### 7.15.3 Member Function Documentation

#### 7.15.3.1 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

#### 7.15.3.2 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

#### 7.15.3.3 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 |

#### 7.15.3.4 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 |

#### 7.15.3.5 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

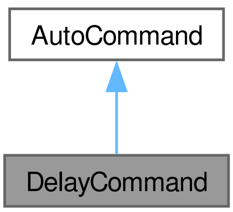
The documentation for this class was generated from the following files:

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

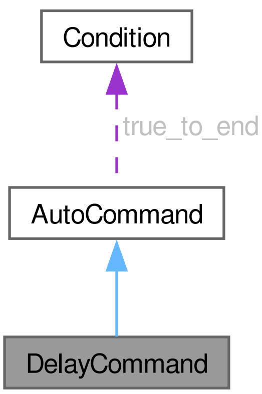
## 7.16 DelayCommand Class Reference

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

Inheritance diagram for DelayCommand:




Collaboration 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

## 7.16.1 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

## 7.16.2 Constructor & Destructor Documentation

### 7.16.2.1 DelayCommand()

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

Construct a delay command

**Parameters**

| | |
|---|---|
| *ms* | the number of milliseconds to delay for |

## 7.16.3 Member Function Documentation

### 7.16.3.1 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.

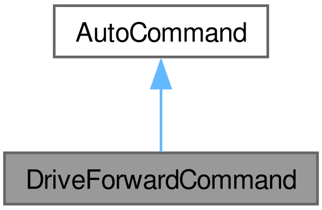
The documentation for this class was generated from the following file:

- include/utils/command_structure/delay_command.h

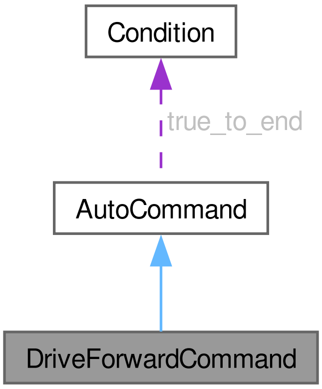
## 7.17 DriveForwardCommand Class Reference

`#include <drive_commands.h>`

Inheritance diagram for DriveForwardCommand:




Collaboration 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

## 7.17.1 Detailed Description

AutoCommand wrapper class for the drive_forward function in the TankDrive class

## 7.17.2 Constructor & Destructor Documentation

### 7.17.2.1 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 |

### 7.17.3 Member Function Documentation

#### 7.17.3.1 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.

#### 7.17.3.2 run()

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

Run drive_forward Overrides run from AutoCommand

**Returns**

true when execution is complete, false otherwise

Reimplemented from AutoCommand.

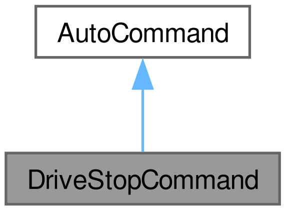
The documentation for this class was generated from the following files:

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

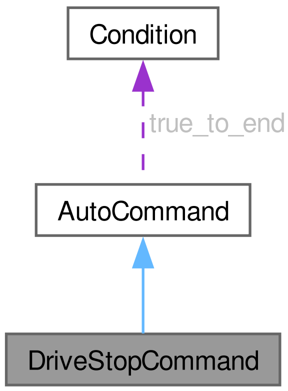
## 7.18 DriveStopCommand Class Reference

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

Inheritance diagram for DriveStopCommand:




Collaboration 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

## 7.18.1 Detailed Description

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

## 7.18.2 Constructor & Destructor Documentation

### 7.18.2.1 DriveStopCommand()

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

Construct a DriveStop Command

**Parameters**

| | |
|---|---|
| *drive_sys* | the drive system we are commanding |

## 7.18.3 Member Function Documentation

### 7.18.3.1 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.

### 7.18.3.2 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.

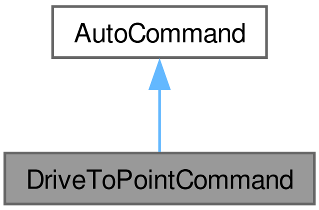
The documentation for this class was generated from the following files:

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

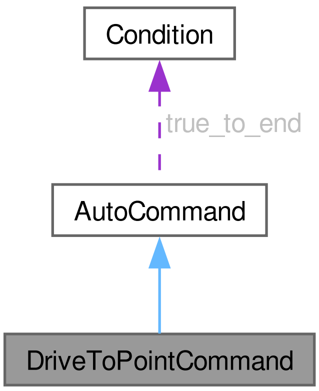
## 7.19 DriveToPointCommand Class Reference

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

Inheritance diagram for DriveToPointCommand:




Collaboration 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

## 7.19.1 Detailed Description

AutoCommand wrapper class for the drive_to_point function in the TankDrive class

## 7.19.2 Constructor & Destructor Documentation

### 7.19.2.1 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 |

#### 7.19.2.2 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 |

### 7.19.3 Member Function Documentation

#### 7.19.3.1 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.

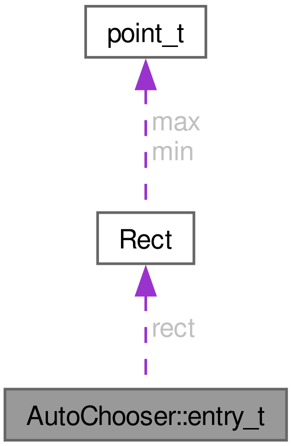
The documentation for this class was generated from the following files:

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

## 7.20 AutoChooser::entry_t Struct Reference

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

Collaboration diagram for AutoChooser::entry_t:




**Public Attributes**

- Rect rect
- std::string name

## 7.20.1 Detailed Description

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

## 7.20.2 Member Data Documentation

### 7.20.2.1 name

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

name of the auto repretsented by the block

### 7.20.2.2 rect

```
Rect AutoChooser::entry_t::rect
```

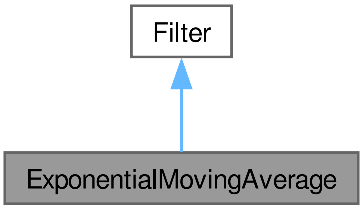
The documentation for this struct was generated from the following file:

- include/utils/auto_chooser.h

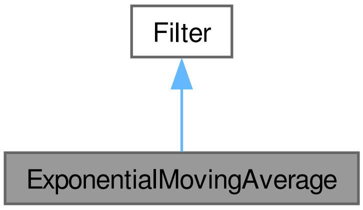
## 7.21 ExponentialMovingAverage Class Reference

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

Inheritance diagram for ExponentialMovingAverage:




Collaboration 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 ()

### 7.21.1 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.

## 7.21.2 Constructor & Destructor Documentation

### 7.21.2.1 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 |

### 7.21.2.2 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 |

## 7.21.3 Member Function Documentation

### 7.21.3.1 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.

### 7.21.3.2 get_size()

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

How many samples the average is made from

**Returns**

the number of samples used to calculate this average

#### 7.21.3.3 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.

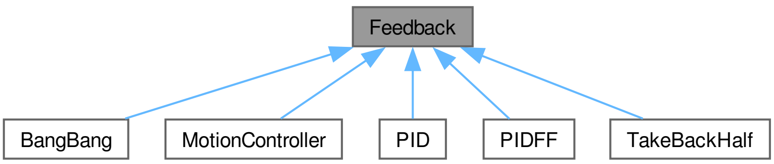
The documentation for this class was generated from the following files:

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

## 7.22 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

### 7.22.1 Detailed Description

Interface so that subsystems can easily switch between feedback loops

**Author**

Ryan McGee

**Date**

9/25/2022

### 7.22.2 Member Function Documentation

#### 7.22.2.1 get()

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

**Returns**

the last saved result from the feedback controller

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

#### 7.22.2.2 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.

#### 7.22.2.3 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.

#### 7.22.2.4 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.

#### 7.22.2.5 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

## 7.23 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.*

## 7.23.1 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

## 7.23.2 Constructor & Destructor Documentation

### 7.23.2.1 FeedForward()

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

Creates a FeedForward object.

**Parameters**

| | |
|---|---|
| *cfg* | Configuration Struct for tuning |

### 7.23.3 Member Function Documentation

#### 7.23.3.1 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

## 7.24 FeedForward::ff_config_t Struct Reference

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

**Public Attributes**

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

### 7.24.1 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

### 7.24.2 Member Data Documentation

#### 7.24.2.1 kA

`double FeedForward::ff_config_t::kA`

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

#### 7.24.2.2 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.

#### 7.24.2.3 kS

`double FeedForward::ff_config_t::kS`

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

#### 7.24.2.4 kV

`double FeedForward::ff_config_t::kV`

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

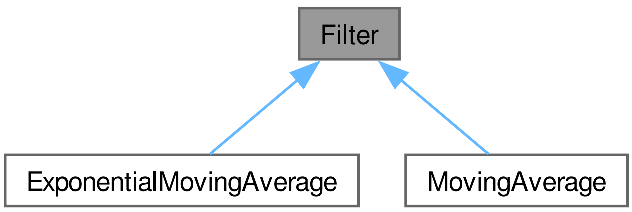
The documentation for this struct was generated from the following file:

- include/utils/controls/feedforward.h

## 7.25 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

### 7.25.1 Detailed Description

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

### 7.25.2 Member Function Documentation

#### 7.25.2.1 add_entry()

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

Implemented in MovingAverage, and ExponentialMovingAverage.

#### 7.25.2.2 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

## 7.26 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)

## 7.26.1 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

## 7.26.2 Constructor & Destructor Documentation

### 7.26.2.1 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 |

## 7.26.3 Member Function Documentation

### 7.26.3.1 get_motors()

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

Returns the motors

**Returns**

the motors used to run the flywheel

#### 7.26.3.2 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

#### 7.26.3.3 getRPM()

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

return the velocity of the flywheel

#### 7.26.3.4 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

#### 7.26.3.5 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()});

#### 7.26.3.6 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 |

#### 7.26.3.7 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 |

#### 7.26.3.8 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

#### 7.26.3.9 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

#### 7.26.3.10 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

### 7.26.4 Friends And Related Symbol Documentation

#### 7.26.4.1 FlywheelPage

```
friend class FlywheelPage [friend]
```

#### 7.26.4.2 spinRPMTask

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

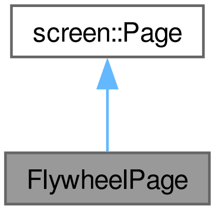
Runs a thread that keeps track of updating flywheel RPM and controlling it accordingly

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

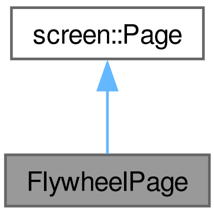
- include/subsystems/flywheel.h
- src/subsystems/flywheel.cpp

## 7.27 FlywheelPage Class Reference

Inheritance diagram for FlywheelPage:




Collaboration 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

### 7.27.1 Constructor & Destructor Documentation

#### 7.27.1.1 FlywheelPage()

```
FlywheelPage::FlywheelPage (
            const Flywheel & fw )  [inline]
```

## 7.27.2 Member Function Documentation

### 7.27.2.1 draw()

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

**See also**

Page::draw

Reimplemented from screen::Page.

### 7.27.2.2 update()

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

**See also**

Page::update

Reimplemented from screen::Page.

## 7.27.3 Member Data Documentation

### 7.27.3.1 window_size

```
const size_t FlywheelPage::window_size = 40 [static]
```

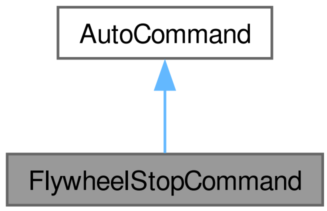
The documentation for this class was generated from the following file:

- src/subsystems/flywheel.cpp

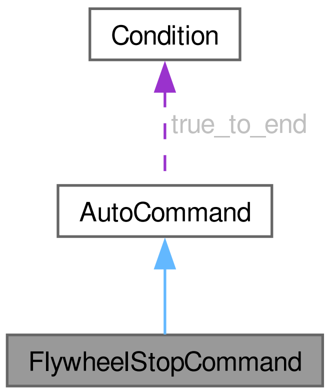
## 7.28 FlywheelStopCommand Class Reference

`#include <flywheel_commands.h>`

Inheritance diagram for FlywheelStopCommand:




Collaboration 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

## 7.28.1 Detailed Description

AutoCommand wrapper class for the stop function in the Flywheel class

## 7.28.2 Constructor & Destructor Documentation

### 7.28.2.1 FlywheelStopCommand()

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

Construct a FlywheelStopCommand

**Parameters**

| | |
|---|---|
| *flywheel* | the flywheel system we are commanding |

## 7.28.3 Member Function Documentation

### 7.28.3.1 run()

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

Run stop Overrides run from AutoCommand

**Returns**

true when execution is complete, false otherwise

Reimplemented from AutoCommand.

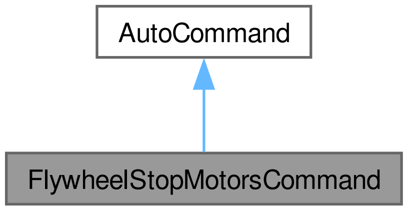
The documentation for this class was generated from the following files:

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

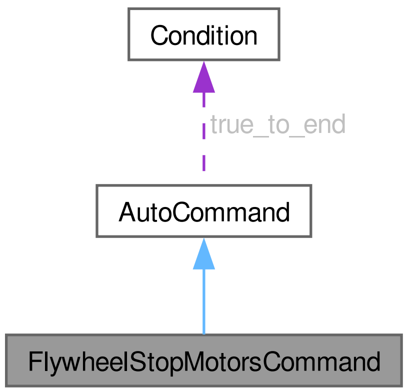
## 7.29 FlywheelStopMotorsCommand Class Reference

`#include <flywheel_commands.h>`

Inheritance diagram for FlywheelStopMotorsCommand:




Collaboration 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

### 7.29.1 Detailed Description

AutoCommand wrapper class for the stopMotors function in the Flywheel class

### 7.29.2 Constructor & Destructor Documentation

#### 7.29.2.1 FlywheelStopMotorsCommand()

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

Construct a FlywheeStopMotors Command

**Parameters**

| | |
|---|---|
| *flywheel* | the flywheel system we are commanding |

### 7.29.3 Member Function Documentation

#### 7.29.3.1 run()

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

Run stop Overrides run from AutoCommand

**Returns**

true when execution is complete, false otherwise

Reimplemented from AutoCommand.

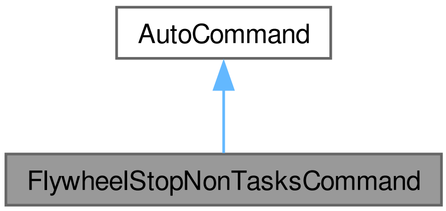
The documentation for this class was generated from the following files:

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

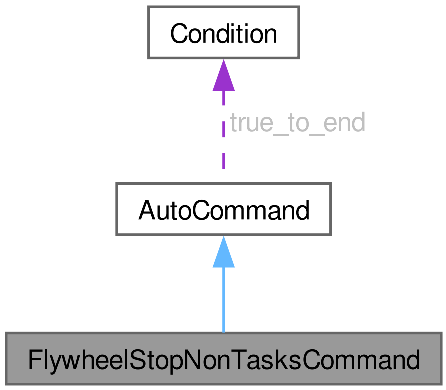
## 7.30 FlywheelStopNonTasksCommand Class Reference

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

Inheritance diagram for FlywheelStopNonTasksCommand:




Collaboration 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

### 7.30.1 Detailed Description

AutoCommand wrapper class for the stopNonTasks function in the Flywheel class

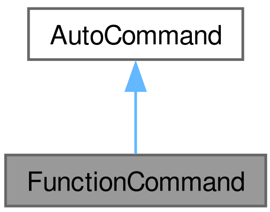
The documentation for this class was generated from the following files:

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

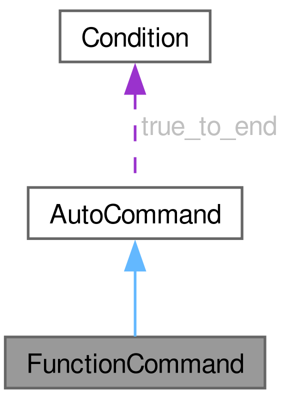
## 7.31 FunctionCommand Class Reference

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

Inheritance diagram for FunctionCommand:




Collaboration 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

## 7.31.1 Detailed Description

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

## 7.31.2 Constructor & Destructor Documentation

### 7.31.2.1 FunctionCommand()

```
FunctionCommand::FunctionCommand (
            std::function< bool(void)> f )  [inline]
```

## 7.31.3 Member Function Documentation

### 7.31.3.1 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:

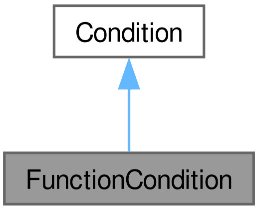
- include/utils/command_structure/auto_command.h

## 7.32 FunctionCondition Class Reference

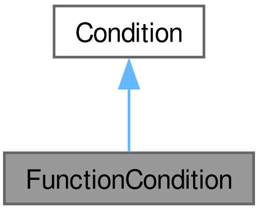
FunctionCondition is a quick and dirty Condition to wrap some expression that should be evaluated at runtime.

`#include <auto_command.h>`

Inheritance diagram for FunctionCondition:




Collaboration 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)

### 7.32.1 Detailed Description

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

### 7.32.2 Constructor & Destructor Documentation

#### 7.32.2.1 FunctionCondition()

```
FunctionCondition::FunctionCondition (
            std::function< bool()> cond,
            std::function< void(void)> timeout = [](){} ) [inline]
```

### 7.32.3 Member Function Documentation

#### 7.32.3.1 test()

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

Implements Condition.

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

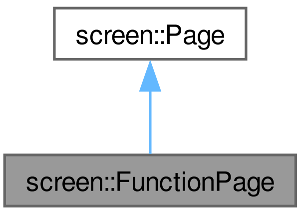
- include/utils/command_structure/auto_command.h
- src/utils/command_structure/auto_command.cpp

## 7.33 screen::FunctionPage Class Reference

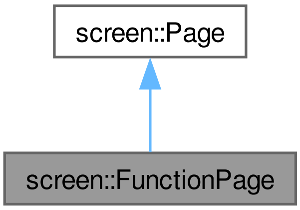
Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything.

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

Inheritance diagram for screen::FunctionPage:



Collaboration 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*

## 7.33.1 Detailed Description

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

## 7.33.2 Constructor & Destructor Documentation

### 7.33.2.1 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 |

### 7.33.3 Member Function Documentation

#### 7.33.3.1 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.

#### 7.33.3.2 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

## 7.34 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)

### 7.34.1 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

### 7.34.2 Member Function Documentation

#### 7.34.2.1 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 |

#### 7.34.2.2 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 |

#### 7.34.2.3 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 |

#### 7.34.2.4 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

## 7.35 GraphDrawer Class Reference

```
#include <graph_drawer.h>
```

### 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)

### 7.35.1 Constructor & Destructor Documentation

#### 7.35.1.1 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 |

## 7.35.2 Member Function Documentation

### 7.35.2.1 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) |

### 7.35.2.2 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 |

### 7.35.2.3 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 |
| *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

## 7.36 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

### 7.36.1 Detailed Description

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

### 7.36.2 Member Function Documentation

#### 7.36.2.1 getPoint()

```
point_t PurePursuit::hermite_point::getPoint ( ) const [inline]
```

#### 7.36.2.2 getTangent()

```
Vector2D PurePursuit::hermite_point::getTangent ( ) const [inline]
```

### 7.36.3 Member Data Documentation

#### 7.36.3.1 dir

```
double PurePursuit::hermite_point::dir
```

#### 7.36.3.2 mag

```
double PurePursuit::hermite_point::mag
```

#### 7.36.3.3 x

```
double PurePursuit::hermite_point::x
```

#### 7.36.3.4 y

```
double PurePursuit::hermite_point::y
```

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

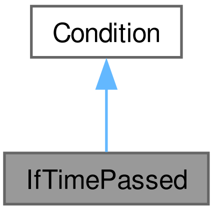
- include/utils/pure_pursuit.h

## 7.37 IfTimePassed Class Reference

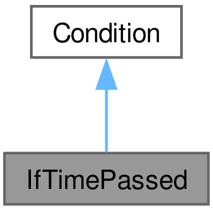
IfTimePassed tests based on time since the command controller was constructed. Returns true if elapsed time > time_s.

`#include <auto_command.h>`

Inheritance diagram for IfTimePassed:




Collaboration 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)

## 7.37.1 Detailed Description

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

## 7.37.2 Constructor & Destructor Documentation

### 7.37.2.1 IfTimePassed()

```
IfTimePassed::IfTimePassed (
            double time_s )
```

## 7.37.3 Member Function Documentation

### 7.37.3.1 test()

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

Implements Condition.

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

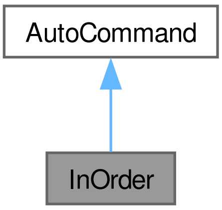
- include/utils/command_structure/auto_command.h
- src/utils/command_structure/auto_command.cpp

## 7.38 InOrder Class Reference

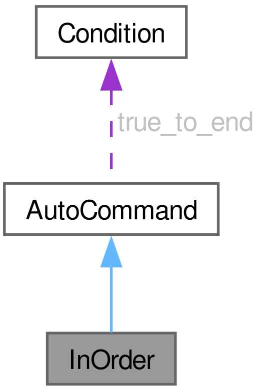
InOrder runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timouts?

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

Inheritance diagram for InOrder:




Collaboration 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

## 7.38.1 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?

## 7.38.2 Constructor & Destructor Documentation

### 7.38.2.1 InOrder() [1/3]

```
InOrder::InOrder (
            const InOrder & other )  [default]
```

### 7.38.2.2 InOrder() [2/3]

```
InOrder::InOrder (
            std::queue< AutoCommand * > cmds )
```

### 7.38.2.3 InOrder() [3/3]

```
InOrder::InOrder (
            std::initializer_list< AutoCommand * > cmds )
```

### 7.38.3 Member Function Documentation

#### 7.38.3.1 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.

#### 7.38.3.2 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

## 7.39 screen::LabelConfig Struct Reference

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

**Public Attributes**

- std::string label

### 7.39.1 Member Data Documentation

#### 7.39.1.1 label

```
std::string screen::LabelConfig::label
```

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

- include/subsystems/screen.h

## 7.40 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))

### 7.40.1 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

### 7.40.2 Constructor & Destructor Documentation

#### 7.40.2.1 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 |

### 7.40.3 Member Function Documentation

#### 7.40.3.1 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 |

#### 7.40.3.2 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 |

#### 7.40.3.3 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). |

#### 7.40.3.4 get_async()

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

**Returns**

whether or not the background thread is running the lift

#### 7.40.3.5 get_setpoint()

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

**Returns**

The current setpoint for the lift

#### 7.40.3.6 hold()

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

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

#### 7.40.3.7 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.

#### 7.40.3.8 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 |

### 7.40.3.9 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

### 7.40.3.10 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 |

### 7.40.3.11 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

### 7.40.3.12 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

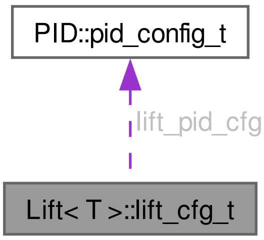
The documentation for this class was generated from the following file:

- include/subsystems/lift.h

## 7.41 Lift< T >::lift_cfg_t Struct Reference

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

Collaboration diagram for Lift< T >::lift_cfg_t:




**Public Attributes**

- double up_speed
- double down_speed
- double softstop_up
- double softstop_down
- PID::pid_config_t lift_pid_cfg

### 7.41.1 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

### 7.41.2 Member Data Documentation

#### 7.41.2.1 down_speed

```
template<typename T >
double Lift< T >::lift_cfg_t::down_speed
```

#### 7.41.2.2 lift_pid_cfg

```
template<typename T >
PID::pid_config_t Lift< T >::lift_cfg_t::lift_pid_cfg
```

#### 7.41.2.3 softstop_down

```
template<typename T >
double Lift< T >::lift_cfg_t::softstop_down
```

#### 7.41.2.4 softstop_up

```
template<typename T >
double Lift< T >::lift_cfg_t::softstop_up
```

#### 7.41.2.5 up_speed

```
template<typename T >
double Lift< T >::lift_cfg_t::up_speed
```

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

- include/subsystems/lift.h

## 7.42 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*

## 7.42.1 Detailed Description

Class to simplify writing to files.

## 7.42.2 Constructor & Destructor Documentation

### 7.42.2.1 Logger() [1/2]

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

Create a logger that will save to a file.

**Parameters**

| | |
|---|---|
| *filename* | the file to save to |

### 7.42.2.2 Logger() [2/2]

```
Logger::Logger (
          const Logger & l )  [delete]
```

copying not allowed

## 7.42.3 Member Function Documentation

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

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

Write a string to the log.

**Parameters**

| | |
|---|---|
| *s* | the string to write |

### 7.42.3.2 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 |

### 7.42.3.3 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 |

### 7.42.3.4 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 |

#### 7.42.3.5 Logln() [1/2]

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

Write a string and newline to the log.

**Parameters**

| | |
|---|---|
| *s* | the string to write |

#### 7.42.3.6 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 |

#### 7.42.3.7 operator=()

```
Logger & Logger::operator= (
            const Logger & l )  [delete]
```

copying not allowed

### 7.42.4 Member Data Documentation

#### 7.42.4.1 MAX_FORMAT_LEN

```
constexpr int Logger::MAX_FORMAT_LEN = 512  [static], [constexpr]
```

maximum size for a string to be before it's written

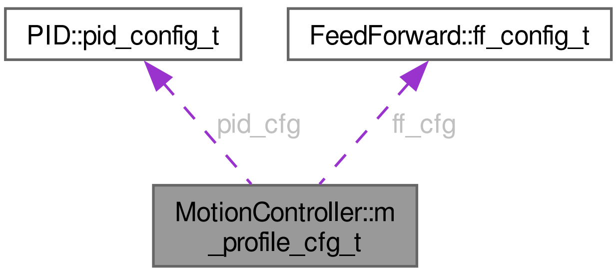
The documentation for this class was generated from the following files:

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

## 7.43 MotionController::m_profile_cfg_t Struct Reference

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

Collaboration diagram for MotionController::m_profile_cfg_t:




### 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*

### 7.43.1 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

### 7.43.2 Member Data Documentation

#### 7.43.2.1 accel

`double MotionController::m_profile_cfg_t::accel`

the most acceleration the robot can do

#### 7.43.2.2 ff_cfg

`FeedForward::ff_config_t MotionController::m_profile_cfg_t::ff_cfg`

configuration parameters for the internal

#### 7.43.2.3 max_v

`double MotionController::m_profile_cfg_t::max_v`

the maximum velocity the robot can drive

#### 7.43.2.4 pid_cfg

`PID::pid_config_t MotionController::m_profile_cfg_t::pid_cfg`

configuration parameters for the internal PID controller

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

- include/utils/controls/motion_controller.h

## 7.44 Mat2 Struct Reference

`#include <geometry.h>`

### 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

### 7.44.1 Member Function Documentation

#### 7.44.1.1 FromRotationDegrees()

```
static Mat2 Mat2::FromRotationDegrees (
             double degrees ) [inline], [static]
```

#### 7.44.1.2 operator*()

```
point_t Mat2::operator* (
             const point_t p ) const [inline]
```

### 7.44.2 Member Data Documentation

#### 7.44.2.1 X11

```
double Mat2::X11
```

#### 7.44.2.2 X12

```
double Mat2::X12
```

#### 7.44.2.3 X21

```
double Mat2::X21
```

#### 7.44.2.4 X22

```
double Mat2::X22
```

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

- include/utils/geometry.h

## 7.45 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)

## 7.45.1 Detailed Description

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

## 7.45.2 Constructor & Destructor Documentation

### 7.45.2.1 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

## 7.45.3 Member Function Documentation

### 7.45.3.1 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.

#### 7.45.3.2 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

#### 7.45.3.3 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) |

#### 7.45.3.4 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:

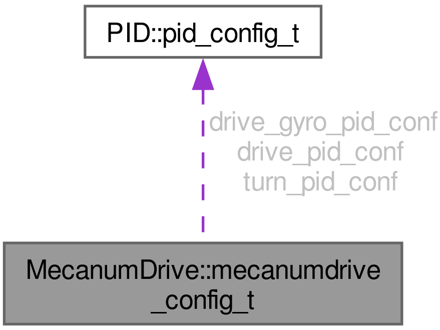
- include/subsystems/mecanum_drive.h

- src/subsystems/mecanum_drive.cpp

## 7.46 MecanumDrive::mecanumdrive_config_t Struct Reference

`#include <mecanum_drive.h>`

Collaboration diagram for MecanumDrive::mecanumdrive_config_t:




**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

### 7.46.1 Detailed Description

Configure the Mecanum drive PID tunings and robot configurations

### 7.46.2 Member Data Documentation

#### 7.46.2.1 drive_gyro_pid_conf

`PID::pid_config_t MecanumDrive::mecanumdrive_config_t::drive_gyro_pid_conf`

#### 7.46.2.2 drive_pid_conf

`PID::pid_config_t MecanumDrive::mecanumdrive_config_t::drive_pid_conf`

#### 7.46.2.3 drive_wheel_diam

```
double MecanumDrive::mecanumdrive_config_t::drive_wheel_diam
```

#### 7.46.2.4 lateral_wheel_diam

```
double MecanumDrive::mecanumdrive_config_t::lateral_wheel_diam
```

#### 7.46.2.5 turn_pid_conf

```
PID::pid_config_t MecanumDrive::mecanumdrive_config_t::turn_pid_conf
```

#### 7.46.2.6 wheelbase_width

```
double MecanumDrive::mecanumdrive_config_t::wheelbase_width
```

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

- include/subsystems/mecanum_drive.h

## 7.47 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*

### 7.47.1 Detailed Description

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

### 7.47.2 Member Data Documentation

#### 7.47.2.1 accel

```
double motion_t::accel
```

1d acceleration at this point in time

#### 7.47.2.2 pos

```
double motion_t::pos
```

1d position at this point in time

#### 7.47.2.3 vel

```
double motion_t::vel
```

1d velocity at this point in time

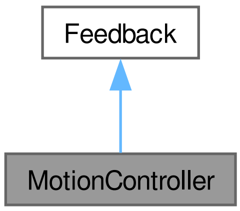
The documentation for this struct was generated from the following file:

- include/utils/controls/trapezoid_profile.h

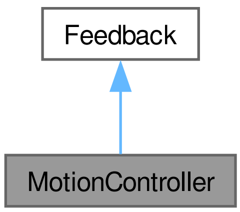
## 7.48 MotionController Class Reference

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

Inheritance diagram for MotionController:




Collaboration 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

## 7.48.1 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

## 7.48.2 Constructor & Destructor Documentation

### 7.48.2.1 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 |

## 7.48.3 Member Function Documentation

### 7.48.3.1 get()

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

**Returns**

the last saved result from the feedback controller

Implements Feedback.

### 7.48.3.2 get_motion()

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

**Returns**

The current postion, velocity and acceleration setpoints

### 7.48.3.3 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.

#### 7.48.3.4 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.

#### 7.48.3.5 Page()

```
screen::Page * MotionController::Page ( )
```

#### 7.48.3.6 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.

#### 7.48.3.7 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

#### 7.48.3.8 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.

### 7.48.4 Friends And Related Symbol Documentation

#### 7.48.4.1 MotionControllerPage

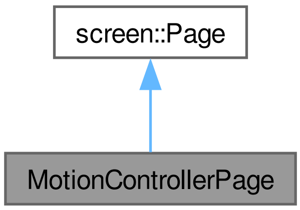
```
friend class MotionControllerPage  [friend]
```

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

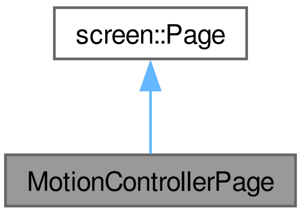
- include/utils/controls/motion_controller.h
- src/utils/controls/motion_controller.cpp

## 7.49 MotionControllerPage Class Reference

Inheritance diagram for MotionControllerPage:




Collaboration 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)*

### 7.49.1 Constructor & Destructor Documentation

#### 7.49.1.1 MotionControllerPage()

```
MotionControllerPage::MotionControllerPage (
            const MotionController & mc )  [inline]
```

## 7.49.2 Member Function Documentation

### 7.49.2.1 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.

### 7.49.2.2 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.

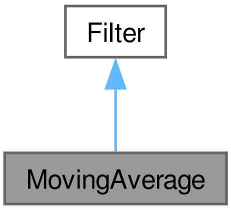
The documentation for this class was generated from the following file:

- src/utils/controls/motion_controller.cpp

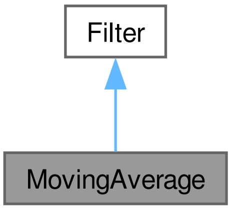
## 7.50 MovingAverage Class Reference

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

Inheritance diagram for MovingAverage:




Collaboration 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

### 7.50.1 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).

## 7.50.2 Constructor & Destructor Documentation

### 7.50.2.1 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 |

### 7.50.2.2 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 |

## 7.50.3 Member Function Documentation

### 7.50.3.1 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.

### 7.50.3.2 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

#### 7.50.3.3 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.

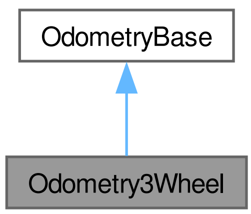
The documentation for this class was generated from the following files:

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

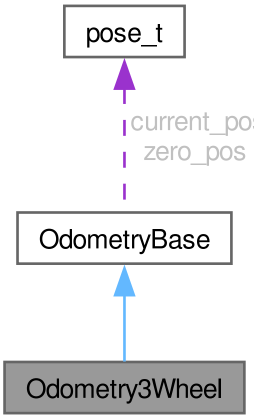
## 7.51 Odometry3Wheel Class Reference

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

Inheritance diagram for Odometry3Wheel:



Collaboration 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

## 7.51.1 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

## 7.51.2 Constructor & Destructor Documentation

### 7.51.2.1 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 |

## 7.51.3 Member Function Documentation

### 7.51.3.1 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

### 7.51.3.2 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

## 7.52 Odometry3Wheel::odometry3wheel_cfg_t Struct Reference

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

#### Public Attributes

- double wheelbase_dist
- double off_axis_center_dist
- double wheel_diam

### 7.52.1 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

### 7.52.2 Member Data Documentation

#### 7.52.2.1 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

#### 7.52.2.2 wheel_diam

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

the diameter of the tracking wheel

#### 7.52.2.3 wheelbase_dist

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

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

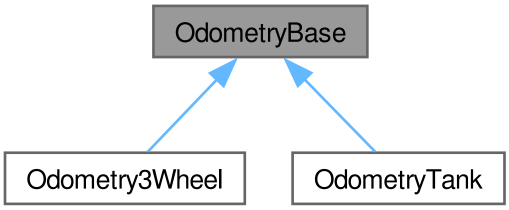
The documentation for this struct was generated from the following file:

- include/subsystems/odometry/odometry_3wheel.h

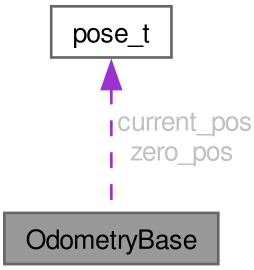
## 7.53 OdometryBase Class Reference

`#include <odometry_base.h>`

Inheritance diagram for OdometryBase:




Collaboration 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

## 7.53.1 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

## 7.53.2 Constructor & Destructor Documentation

### 7.53.2.1 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 |

### 7.53.3 Member Function Documentation

#### 7.53.3.1 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.

#### 7.53.3.2 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.

#### 7.53.3.3 get_accel()

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

Get the current acceleration

**Returns**

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

#### 7.53.3.4 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)

#### 7.53.3.5 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)

#### 7.53.3.6 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

#### 7.53.3.7 get_speed()

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

Get the current speed

**Returns**

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

#### 7.53.3.8 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

#### 7.53.3.9 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

#### 7.53.3.10 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.

#### 7.53.3.11 SetPositionCmd()

```
AutoCommand * OdometryBase::SetPositionCmd (
            const pose_t & newpos = zero_pos )
```

#### 7.53.3.12 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.

#### 7.53.3.13 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.

### 7.53.4 Member Data Documentation

#### 7.53.4.1 accel

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

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

#### 7.53.4.2 ang_accel_deg

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

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

#### 7.53.4.3 ang_speed_deg

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

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

#### 7.53.4.4 current_pos

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

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

#### 7.53.4.5 end_task

```
bool OdometryBase::end_task = false
```

end_task is true if we instruct the odometry thread to shut down

#### 7.53.4.6 handle

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

handle to the vex task that is running the odometry code

#### 7.53.4.7 mut

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

Mutex to control multithreading

#### 7.53.4.8 speed

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

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

#### 7.53.4.9 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:

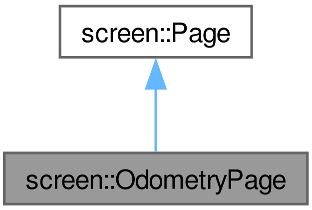
- include/subsystems/odometry/odometry_base.h
- src/subsystems/odometry/odometry_base.cpp

## 7.54 screen::OdometryPage Class Reference

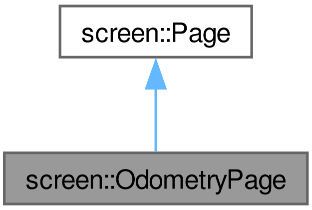
a page that shows odometry position and rotation and a map (if an sd card with the file is on)

`#include <screen.h>`

Inheritance diagram for screen::OdometryPage:




Collaboration 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

### 7.54.1 Detailed Description

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

## 7.54.2 Constructor & Destructor Documentation

### 7.54.2.1 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 |

## 7.54.3 Member Function Documentation

### 7.54.3.1 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.

### 7.54.3.2 update()

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

**See also**

Page::update

Reimplemented from screen::Page.

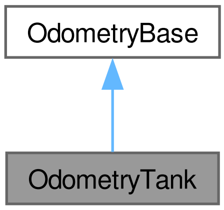
The documentation for this class was generated from the following files:

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

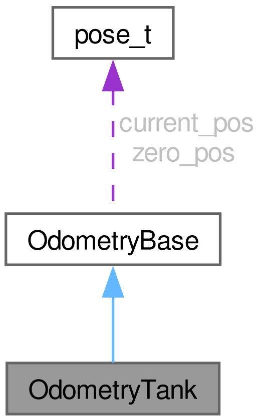
## 7.55 OdometryTank Class Reference

`#include <odometry_tank.h>`

Inheritance diagram for OdometryTank:




Collaboration 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

## 7.55.1 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.

## 7.55.2 Constructor & Destructor Documentation

### 7.55.2.1 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(). |

#### 7.55.2.2 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(). |

#### 7.55.2.3 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(). |

### 7.55.3 Member Function Documentation

#### 7.55.3.1 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.

#### 7.55.3.2 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.

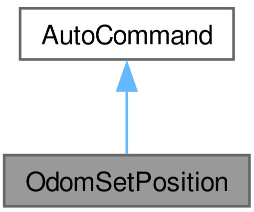
The documentation for this class was generated from the following files:

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

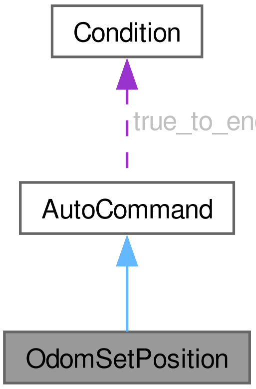
## 7.56 OdomSetPosition Class Reference

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

Inheritance diagram for OdomSetPosition:



Collaboration 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

## 7.56.1 Detailed Description

AutoCommand wrapper class for the set_position function in the Odometry class

## 7.56.2 Constructor & Destructor Documentation

### 7.56.2.1 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 |

### 7.56.3 Member Function Documentation

#### 7.56.3.1 run()

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

Run set_position Overrides run from AutoCommand

**Returns**

true when execution is complete, false otherwise

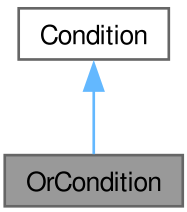
Reimplemented from AutoCommand.

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

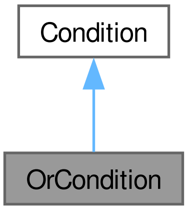
- include/utils/command_structure/drive_commands.h
- src/utils/command_structure/drive_commands.cpp

## 7.57 OrCondition Class Reference

Inheritance diagram for OrCondition:



Collaboration 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)

## 7.57.1 Constructor & Destructor Documentation

### 7.57.1.1 OrCondition()

```
OrCondition::OrCondition (
            Condition * A,
            Condition * B )  [inline]
```

## 7.57.2 Member Function Documentation

### 7.57.2.1 test()

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

Implements Condition.

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

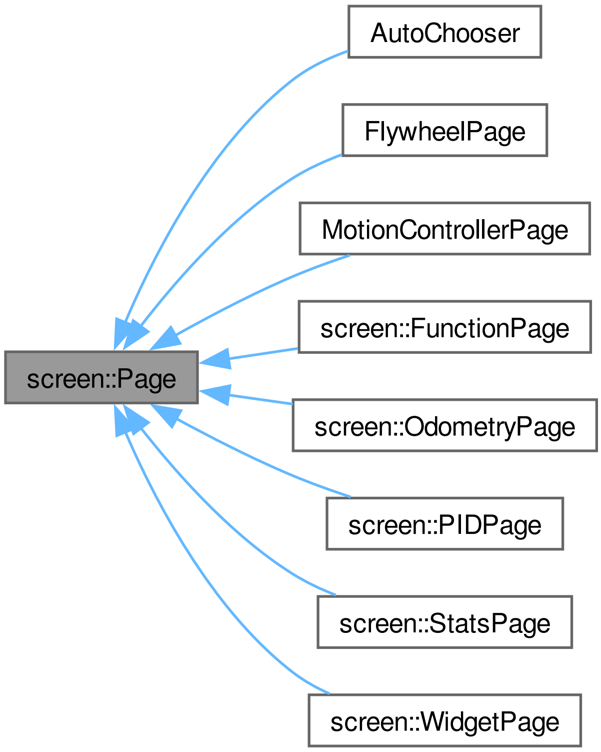
- src/utils/command_structure/auto_command.cpp

## 7.58 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)*

### 7.58.1 Detailed Description

Page describes one part of the screen slideshow.

### 7.58.2 Member Function Documentation

#### 7.58.2.1 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.

#### 7.58.2.2 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:

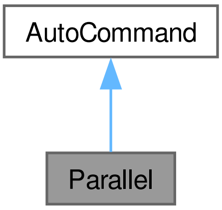
- include/subsystems/screen.h

## 7.59 Parallel Class Reference

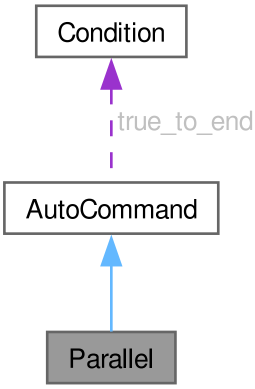
Parallel runs multiple commands in parallel and waits for all to finish before continuing. if none finish before this command's timeout, it will call on_timeout on all children continue.

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

Inheritance diagram for Parallel:




Collaboration 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

## 7.59.1 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.

## 7.59.2 Constructor & Destructor Documentation

### 7.59.2.1 Parallel()

```
Parallel::Parallel (
            std::initializer_list< AutoCommand * > cmds )
```

## 7.59.3 Member Function Documentation

### 7.59.3.1 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.

### 7.59.3.2 run()

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

Executes the command Overridden by child classes

**Returns**

true when the command is finished, false otherwise

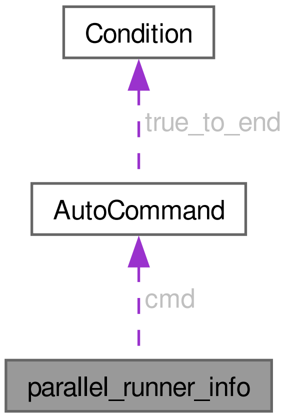
Reimplemented from AutoCommand.

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

## 7.60 parallel_runner_info Struct Reference

Collaboration diagram for parallel_runner_info:




### Public Attributes

- int index
- std::vector< vex::task * > * runners
- AutoCommand * cmd

### 7.60.1 Member Data Documentation

#### 7.60.1.1 cmd

```
AutoCommand* parallel_runner_info::cmd
```

#### 7.60.1.2 index

```
int parallel_runner_info::index
```

#### 7.60.1.3 runners

```
std::vector<vex::task *>* parallel_runner_info::runners
```

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

- src/utils/command_structure/auto_command.cpp

## 7.61 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 ()

### 7.61.1 Detailed Description

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

### 7.61.2 Constructor & Destructor Documentation

#### 7.61.2.1 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 |

### 7.61.3 Member Function Documentation

#### 7.61.3.1 get_points()

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

Get the points associated with this Path

#### 7.61.3.2 get_radius()

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

Get the radius associated with this Path

#### 7.61.3.3 is_valid()

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

Get whether this path will behave as expected

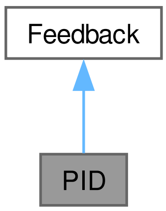
The documentation for this class was generated from the following files:

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

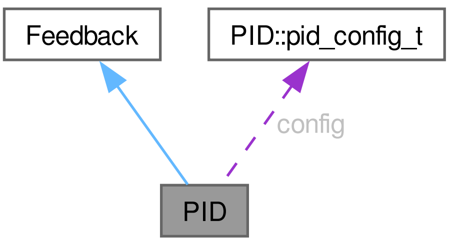
## 7.62 PID Class Reference

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

Inheritance diagram for PID:




Collaboration 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

## 7.62.1 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

## 7.62.2 Member Enumeration Documentation

### 7.62.2.1 ERROR_TYPE

`enum PID::ERROR_TYPE`

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

**Enumerator**

| | |
|---|---|
| LINEAR | |
| ANGULAR | |

## 7.62.3 Constructor & Destructor Documentation

### 7.62.3.1 PID()

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

Create the PID object

**Parameters**

| | |
|---|---|
| *config* | the configuration data for this controller |

Create the PID object

## 7.62.4 Member Function Documentation

### 7.62.4.1 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.

### 7.62.4.2 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

#### 7.62.4.3 get_sensor_val()

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

gets the sensor value that we were last updated with

**Returns**

sensor_val

#### 7.62.4.4 get_target()

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

Get the PID's target

**Returns**

the target the PID controller is trying to achieve

#### 7.62.4.5 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.

#### 7.62.4.6 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.

#### 7.62.4.7 reset()

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

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

#### 7.62.4.8 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.

#### 7.62.4.9 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

#### 7.62.4.10 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.

### 7.62.5 Member Data Documentation

#### 7.62.5.1 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

## 7.63 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

### 7.63.1 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

### 7.63.2 Member Data Documentation

#### 7.63.2.1 d

```
double PID::pid_config_t::d
```

derivitave coeffecient d $*$ derivative(error)

#### 7.63.2.2 deadband

```
double PID::pid_config_t::deadband
```

at what threshold are we close enough to be finished

#### 7.63.2.3 error_method

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

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

#### 7.63.2.4 i

```
double PID::pid_config_t::i
```

integral coeffecient i $*$ integral(error)

#### 7.63.2.5 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

#### 7.63.2.6 p

```
double PID::pid_config_t::p
```

proportional coeffecient p ∗ error()

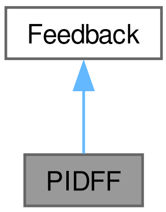
The documentation for this struct was generated from the following file:

- include/utils/controls/pid.h

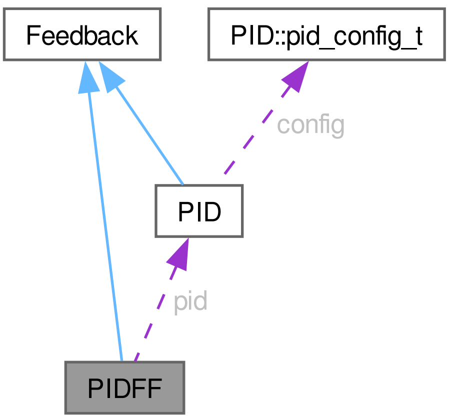
## 7.64 PIDFF Class Reference

```
#include <pidff.h>
```

Inheritance diagram for PIDFF:




Collaboration 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

## 7.64.1 Constructor & Destructor Documentation

### 7.64.1.1 PIDFF()

```
PIDFF::PIDFF (
            PID::pid_config_t & pid_cfg,
            FeedForward::ff_config_t & ff_cfg )
```

## 7.64.2 Member Function Documentation

### 7.64.2.1 get()

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

**Returns**

the last saved result from the feedback controller

Implements Feedback.

### 7.64.2.2 get_sensor_val()

```
double PIDFF::get_sensor_val ( ) const
```

### 7.64.2.3 get_target()

```
double PIDFF::get_target ( ) const
```

### 7.64.2.4 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.

#### 7.64.2.5 is_on_target()

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

**Returns**

true if the feedback controller has reached it's setpoint

Implements Feedback.

#### 7.64.2.6 reset()

```
void PIDFF::reset ( )
```

#### 7.64.2.7 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.

#### 7.64.2.8 set_target()

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

Set the target of the PID loop

**Parameters**

| | |
|---|---|
| *set←_pt* | Setpoint / target value |

#### 7.64.2.9 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.

#### 7.64.2.10 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

### 7.64.3 Member Data Documentation

#### 7.64.3.1 pid

```
PID PIDFF::pid
```

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

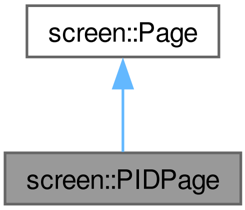
- include/utils/controls/pidff.h
- src/utils/controls/pidff.cpp

## 7.65 screen::PIDPage Class Reference

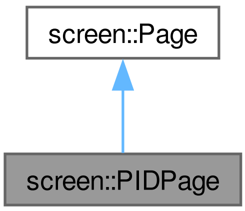
PIDPage provides a way to tune a pid controller on the screen.

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

Inheritance diagram for screen::PIDPage:




Collaboration 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

## 7.65.1 Detailed Description

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

## 7.65.2 Constructor & Destructor Documentation

### 7.65.2.1 PIDPage() [1/2]

```
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 |

### 7.65.2.2 PIDPage() [2/2]

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

## 7.65.3 Member Function Documentation

### 7.65.3.1 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.

#### 7.65.3.2 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

## 7.66 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*

### 7.66.1 Detailed Description

Data structure representing an X,Y coordinate

### 7.66.2 Member Function Documentation

#### 7.66.2.1 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

#### 7.66.2.2 operator*()

```
point_t point_t::operator* (
            double s ) const  [inline]
```

#### 7.66.2.3 operator+() [1/2]

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

#### 7.66.2.4 operator+() [2/2]

```
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)

#### 7.66.2.5 operator-() [1/2]

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

#### 7.66.2.6 operator-() [2/2]

```
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)

#### 7.66.2.7 operator/()

```
point_t point_t::operator/ (
            double s ) const  [inline]
```

#### 7.66.2.8 operator==()

```
bool point_t::operator== (
            const point_t & rhs )  [inline]
```

### 7.66.3 Member Data Documentation

#### 7.66.3.1 x

```
double point_t::x
```

the x position in space

#### 7.66.3.2 y

```
double point_t::y
```

the y position in space

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

- include/utils/geometry.h

## 7.67 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*

### 7.67.1 Detailed Description

Describes a single position and rotation

### 7.67.2 Member Function Documentation

#### 7.67.2.1 get_point()

```
point_t pose_t::get_point ( )  [inline]
```

### 7.67.3 Member Data Documentation

#### 7.67.3.1 rot

```
double pose_t::rot
```

rotation in the world

#### 7.67.3.2 x

```
double pose_t::x
```

x position in the world

#### 7.67.3.3 y

```
double pose_t::y
```

y position in the world

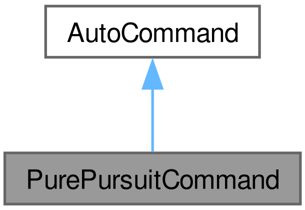
The documentation for this struct was generated from the following file:

- include/utils/geometry.h

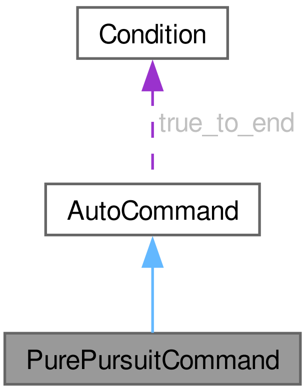
## 7.68 PurePursuitCommand Class Reference

`#include <drive_commands.h>`

Inheritance diagram for PurePursuitCommand:




Collaboration 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

## 7.68.1 Detailed Description

Autocommand wrapper class for pure pursuit function in the TankDrive class

## 7.68.2 Constructor & Destructor Documentation

### 7.68.2.1 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) |

## 7.68.3 Member Function Documentation

### 7.68.3.1 on_timeout()

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

Reset the drive system when it times out

Reimplemented from AutoCommand.

#### 7.68.3.2 run()

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

Direct call to TankDrive::pure_pursuit

Reimplemented from AutoCommand.

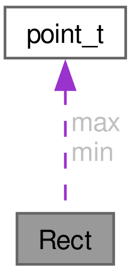
The documentation for this class was generated from the following files:

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

## 7.69 Rect Struct Reference

```
#include <geometry.h>
```

Collaboration diagram for Rect:




### 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

## 7.69.1 Member Function Documentation

### 7.69.1.1 center()

```
point_t Rect::center ( ) const  [inline]
```

### 7.69.1.2 contains()

```
bool Rect::contains (
            point_t p ) const  [inline]
```

### 7.69.1.3 dimensions()

```
point_t Rect::dimensions ( ) const  [inline]
```

### 7.69.1.4 from_min_and_size()

```
static Rect Rect::from_min_and_size (
            point_t min,
            point_t size )  [inline], [static]
```

### 7.69.1.5 height()

```
double Rect::height ( ) const  [inline]
```

### 7.69.1.6 width()

```
double Rect::width ( ) const  [inline]
```

## 7.69.2 Member Data Documentation

### 7.69.2.1 max

```
point_t Rect::max
```

### 7.69.2.2 min

```
point_t Rect::min
```

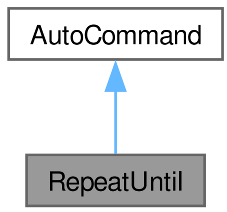
The documentation for this struct was generated from the following file:

- include/utils/geometry.h

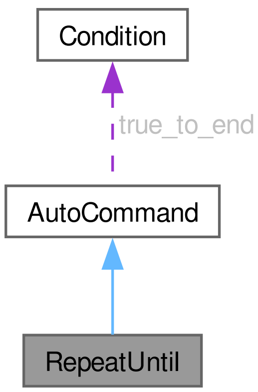
## 7.70 RepeatUntil Class Reference

`#include <auto_command.h>`

Inheritance diagram for RepeatUntil:




Collaboration 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

## 7.70.1 Constructor & Destructor Documentation

### 7.70.1.1 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 |

### 7.70.1.2 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 |

### 7.70.2 Member Function Documentation

#### 7.70.2.1 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.

#### 7.70.2.2 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.

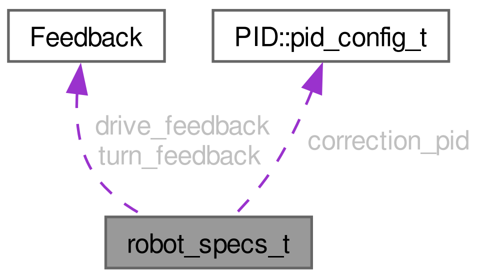
The documentation for this class was generated from the following files:

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

## 7.71 robot_specs_t Struct Reference

```
#include <robot_specs.h>
```

Collaboration diagram for robot_specs_t:



**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*

### 7.71.1 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.

### 7.71.2 Member Data Documentation

#### 7.71.2.1 correction_pid

```
PID::pid_config_t robot_specs_t::correction_pid
```

the pid controller to keep the robot driving in as straight a line as possible

#### 7.71.2.2 dist_between_wheels

```
double robot_specs_t::dist_between_wheels
```

the distance between centers of the central drive wheels

#### 7.71.2.3 drive_correction_cutoff

```
double robot_specs_t::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

#### 7.71.2.4 drive_feedback

```
Feedback* robot_specs_t::drive_feedback
```

the default feedback for autonomous driving

#### 7.71.2.5 odom_gear_ratio

```
double robot_specs_t::odom_gear_ratio
```

the ratio of the odometry wheel to the encoder reading odometry data

#### 7.71.2.6 odom_wheel_diam

```
double robot_specs_t::odom_wheel_diam
```

the diameter of the wheels used for

#### 7.71.2.7 robot_radius

```
double robot_specs_t::robot_radius
```

if you were to draw a circle with this radius, the robot would be entirely contained within it

#### 7.71.2.8 turn_feedback

```
Feedback* robot_specs_t::turn_feedback
```

the defualt feedback for autonomous turning

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

- include/robot_specs.h

## 7.72 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

### 7.72.1 Detailed Description

The ScreenData class holds the data that will be passed to the screen thread you probably shouldnt have to use it.

### 7.72.2 Constructor & Destructor Documentation

#### 7.72.2.1 ScreenData()

```
screen::ScreenData::ScreenData (
            const std::vector< Page * > & m_pages,
            int m_page,
            vex::brain::lcd & m_screen ) [inline]
```

### 7.72.3 Member Data Documentation

#### 7.72.3.1 page

```
int screen::ScreenData::page = 0
```

#### 7.72.3.2 pages

```
std::vector<Page *> screen::ScreenData::pages
```

#### 7.72.3.3 screen

```
vex::brain::lcd screen::ScreenData::screen
```

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

- src/subsystems/screen.cpp

## 7.73 screen::ScreenRect Struct Reference

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

#### Public Attributes

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

### 7.73.1 Member Data Documentation

#### 7.73.1.1 x1

`uint32_t screen::ScreenRect::x1`

#### 7.73.1.2 x2

`uint32_t screen::ScreenRect::x2`

#### 7.73.1.3 y1

`uint32_t screen::ScreenRect::y1`

#### 7.73.1.4 y2

`uint32_t screen::ScreenRect::y2`

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

- include/subsystems/screen.h

## 7.74 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*

## 7.74.1 Detailed Description

Serializes Arbitrary data to a file on the SD Card.

## 7.74.2 Constructor & Destructor Documentation

### 7.74.2.1 ∼Serializer()

```
Serializer::∼Serializer ( ) [inline]
```

Save and close upon destruction (bc of vex, this doesnt always get called when the program ends. To be sure, call save_to_disk)

### 7.74.2.2 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 |

## 7.74.3 Member Function Documentation

### 7.74.3.1 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

#### 7.74.3.2 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

#### 7.74.3.3 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

#### 7.74.3.4 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

#### 7.74.3.5 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 |

#### 7.74.3.6 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 |

#### 7.74.3.7 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 |

#### 7.74.3.8 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 |

#### 7.74.3.9 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

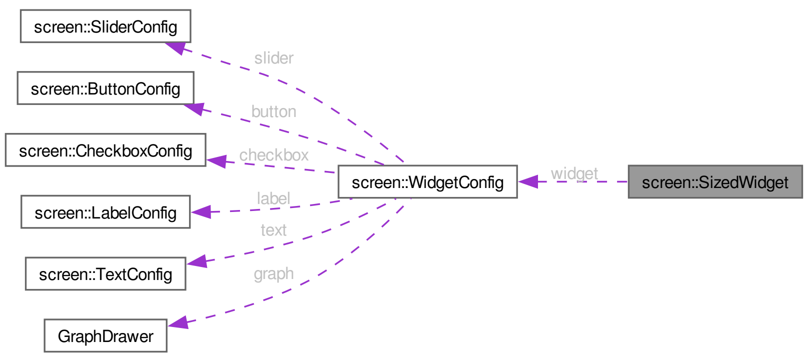
The documentation for this class was generated from the following files:

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

## 7.75 screen::SizedWidget Struct Reference

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

Collaboration diagram for screen::SizedWidget:




### Public Attributes

- int size
- WidgetConfig & widget

### 7.75.1 Member Data Documentation

#### 7.75.1.1 size

```
int screen::SizedWidget::size
```

#### 7.75.1.2 widget

```
WidgetConfig& screen::SizedWidget::widget
```

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

- include/subsystems/screen.h

## 7.76 SliderCfg Struct Reference

```
#include <layout.h>
```

**Public Attributes**

- double & val
- double min
- double max

### 7.76.1 Member Data Documentation

#### 7.76.1.1 max

```
double SliderCfg::max
```

#### 7.76.1.2 min

```
double SliderCfg::min
```

#### 7.76.1.3 val

```
double& SliderCfg::val
```

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

- include/subsystems/layout.h

## 7.77 screen::SliderConfig Struct Reference

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

### Public Attributes

- double & val
- double low
- double high

### 7.77.1 Member Data Documentation

#### 7.77.1.1 high

```
double screen::SliderConfig::high
```

#### 7.77.1.2 low

```
double screen::SliderConfig::low
```

#### 7.77.1.3 val

```
double& screen::SliderConfig::val
```

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

- include/subsystems/screen.h

## 7.78 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*

## 7.78.1 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.

## 7.78.2 Constructor & Destructor Documentation

### 7.78.2.1 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 |

## 7.78.3 Member Function Documentation

### 7.78.3.1 draw()

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

Page::draws the slide to the screen

### 7.78.3.2 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

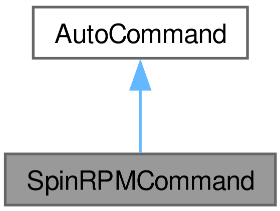
The documentation for this class was generated from the following files:

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

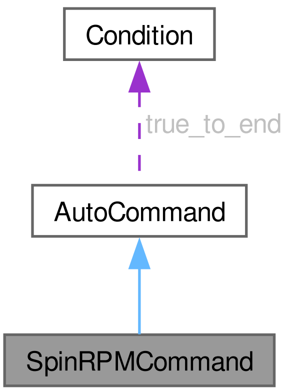
## 7.79 SpinRPMCommand Class Reference

`#include <flywheel_commands.h>`

Inheritance diagram for SpinRPMCommand:




Collaboration 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

## 7.79.1 Detailed Description

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

## 7.79.2 Constructor & Destructor Documentation

### 7.79.2.1 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]

### 7.79.3 Member Function Documentation

#### 7.79.3.1 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

## 7.80 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

### 7.80.1 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.

### 7.80.2 Member Function Documentation

#### 7.80.2.1 getY()

```
double PurePursuit::spline::getY (
            double x )  [inline]
```

### 7.80.3 Member Data Documentation

#### 7.80.3.1 a

```
double PurePursuit::spline::a
```

#### 7.80.3.2 b

```
double PurePursuit::spline::b
```

#### 7.80.3.3 c

```
double PurePursuit::spline::c
```

#### 7.80.3.4 d

```
double PurePursuit::spline::d
```

#### 7.80.3.5 x_end

```
double PurePursuit::spline::x_end
```

#### 7.80.3.6 x_start

```
double PurePursuit::spline::x_start
```

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

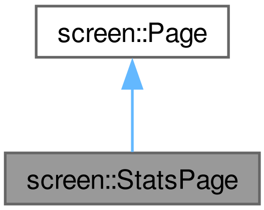
- include/utils/pure_pursuit.h

## 7.81 screen::StatsPage Class Reference

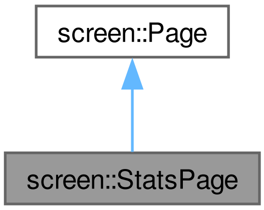
Draws motor stats and battery stats to the screen.

`#include <screen.h>`

Inheritance diagram for screen::StatsPage:




Collaboration 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

### 7.81.1 Detailed Description

Draws motor stats and battery stats to the screen.

### 7.81.2 Constructor & Destructor Documentation

#### 7.81.2.1 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 |

## 7.81.3 Member Function Documentation

### 7.81.3.1 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.

### 7.81.3.2 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:

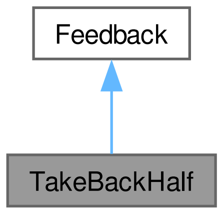
- include/subsystems/screen.h
- src/subsystems/screen.cpp

## 7.82 TakeBackHalf Class Reference

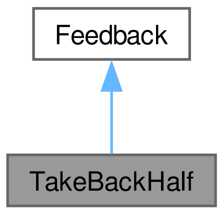
A velocity controller.

`#include <take_back_half.h>`

Inheritance diagram for TakeBackHalf:




Collaboration 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

### 7.82.1 Detailed Description

A velocity controller.

**Warning**

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

### 7.82.2 Constructor & Destructor Documentation

#### 7.82.2.1 TakeBackHalf()

```
TakeBackHalf::TakeBackHalf (
            double TBH_gain,
            double first_cross_split,
            double on_target_threshold )
```

### 7.82.3 Member Function Documentation

#### 7.82.3.1 get()

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

**Returns**

the last saved result from the feedback controller

Implements Feedback.

#### 7.82.3.2 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.

#### 7.82.3.3 is_on_target()

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

**Returns**

true if the feedback controller has reached it's setpoint

Implements Feedback.

#### 7.82.3.4 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.

#### 7.82.3.5 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.

### 7.82.4 Member Data Documentation

#### 7.82.4.1 first_cross_split

```
double TakeBackHalf::first_cross_split
```

#### 7.82.4.2 TBH_gain

```
double TakeBackHalf::TBH_gain
```

tuned parameter

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

## 7.83 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)

## 7.83.1 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

## 7.83.2 Member Enumeration Documentation

### 7.83.2.1 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 |

## 7.83.3 Constructor & Destructor Documentation

### 7.83.3.1 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 |

### 7.83.4 Member Function Documentation

#### 7.83.4.1 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

#### 7.83.4.2 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

#### 7.83.4.3 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 |

#### 7.83.4.4 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 |

#### 7.83.4.5 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) |

#### 7.83.4.6 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

#### 7.83.4.7 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

#### 7.83.4.8 DriveForwardCmd() [1/2]

```
AutoCommand * TankDrive::DriveForwardCmd (
            double dist,
            vex::directionType dir = vex::forward,
            double max_speed = 1.0,
            double end_speed = 0.0 )
```

#### 7.83.4.9 DriveForwardCmd() [2/2]

```
AutoCommand * TankDrive::DriveForwardCmd (
            Feedback & fb,
            double dist,
            vex::directionType dir = vex::forward,
            double max_speed = 1.0,
            double end_speed = 0.0 )
```

#### 7.83.4.10 DriveToPointCmd() [1/2]

```
AutoCommand * TankDrive::DriveToPointCmd (
            Feedback & fb,
            point_t pt,
            vex::directionType dir = vex::forward,
            double max_speed = 1.0,
            double end_speed = 0.0 )
```

#### 7.83.4.11 DriveToPointCmd() [2/2]

```
AutoCommand * TankDrive::DriveToPointCmd (
            point_t pt,
            vex::directionType dir = vex::forward,
            double max_speed = 1.0,
            double end_speed = 0.0 )
```

#### 7.83.4.12 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

#### 7.83.4.13 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

#### 7.83.4.14 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

#### 7.83.4.15 PurePursuitCmd() [1/2]

```
AutoCommand * TankDrive::PurePursuitCmd (
            Feedback & feedback,
            PurePursuit::Path path,
            directionType dir,
            double max_speed = 1,
            double end_speed = 0 )
```

#### 7.83.4.16 PurePursuitCmd() [2/2]

```
AutoCommand * TankDrive::PurePursuitCmd (
            PurePursuit::Path path,
            directionType dir,
            double max_speed = 1,
            double end_speed = 0 )
```

#### 7.83.4.17 reset_auto()

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

Reset the initialization for autonomous drive functions

#### 7.83.4.18 stop()

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

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

#### 7.83.4.19 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

#### 7.83.4.20 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

#### 7.83.4.21 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

#### 7.83.4.22 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

#### 7.83.4.23 TurnDegreesCmd() [1/2]

```
AutoCommand * TankDrive::TurnDegreesCmd (
            double degrees,
            double max_speed = 1.0,
            double start_speed = 0.0 )
```

#### 7.83.4.24 TurnDegreesCmd() [2/2]

```
AutoCommand * TankDrive::TurnDegreesCmd (
            Feedback & fb,
            double degrees,
            double max_speed = 1.0,
            double end_speed = 0.0 )
```

#### 7.83.4.25 TurnToHeadingCmd() [1/2]

```
AutoCommand * TankDrive::TurnToHeadingCmd (
            double heading,
            double max_speed = 1.0,
            double end_speed = 0.0 )
```

#### 7.83.4.26 TurnToHeadingCmd() [2/2]

```
AutoCommand * TankDrive::TurnToHeadingCmd (
            Feedback & fb,
            double heading,
            double max_speed = 1.0,
            double end_speed = 0.0 )
```

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

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

## 7.84 screen::TextConfig Struct Reference

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

### Public Attributes

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

### 7.84.1 Member Data Documentation

#### 7.84.1.1 text

```
std::function<std::string()> screen::TextConfig::text
```

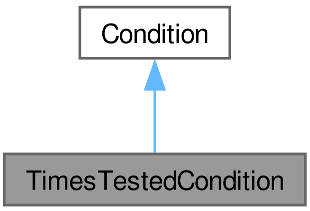
The documentation for this struct was generated from the following file:

- include/subsystems/screen.h

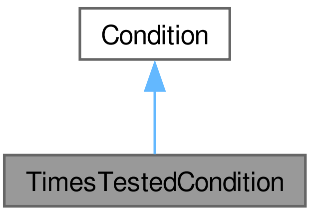
## 7.85 TimesTestedCondition Class Reference

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

Inheritance diagram for TimesTestedCondition:




Collaboration 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)

## 7.85.1 Constructor & Destructor Documentation

### 7.85.1.1 TimesTestedCondition()

```
TimesTestedCondition::TimesTestedCondition (
            size_t N )  [inline]
```

## 7.85.2 Member Function Documentation

### 7.85.2.1 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

# 7.86 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*

### 7.86.1 Detailed Description

trapezoid_profile_segment_t is a description of one constant acceleration segment of a trapezoid motion profile

### 7.86.2 Member Data Documentation

#### 7.86.2.1 accel

```
double trapezoid_profile_segment_t::accel
```

1d acceleration during the segment

#### 7.86.2.2 duration

```
double trapezoid_profile_segment_t::duration
```

duration of the segment

#### 7.86.2.3 pos_after

```
double trapezoid_profile_segment_t::pos_after
```

1d position after this segment concludes

#### 7.86.2.4 vel_after

```
double trapezoid_profile_segment_t::vel_after
```

1d velocity after this segment concludes

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

- include/utils/controls/trapezoid_profile.h

## 7.87 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

## 7.87.1 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

## 7.87.2 Constructor & Destructor Documentation

### 7.87.2.1 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 |

## 7.87.3 Member Function Documentation

### 7.87.3.1 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

### 7.87.3.2 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

#### 7.87.3.3 get_accel()

```
double TrapezoidProfile::get_accel ( ) const
```

#### 7.87.3.4 get_max_v()

```
double TrapezoidProfile::get_max_v ( ) const
```

#### 7.87.3.5 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

#### 7.87.3.6 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 |

#### 7.87.3.7 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 |

#### 7.87.3.8 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 |

#### 7.87.3.9 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 |

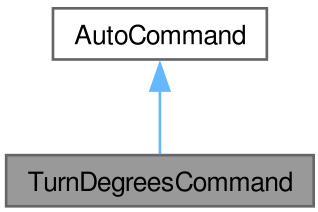
The documentation for this class was generated from the following files:

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

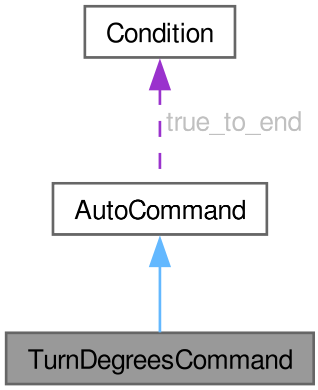
## 7.88 TurnDegreesCommand Class Reference

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

Inheritance diagram for TurnDegreesCommand:



Collaboration 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

### 7.88.1 Detailed Description

AutoCommand wrapper class for the turn_degrees function in the TankDrive class

## 7.88.2 Constructor & Destructor Documentation

### 7.88.2.1 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 |

## 7.88.3 Member Function Documentation

### 7.88.3.1 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.

### 7.88.3.2 run()

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

Run turn_degrees Overrides run from AutoCommand

**Returns**

true when execution is complete, false otherwise

Reimplemented from AutoCommand.

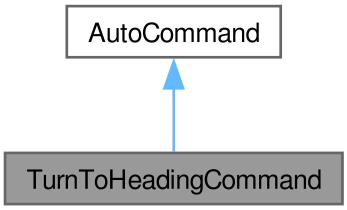
The documentation for this class was generated from the following files:

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

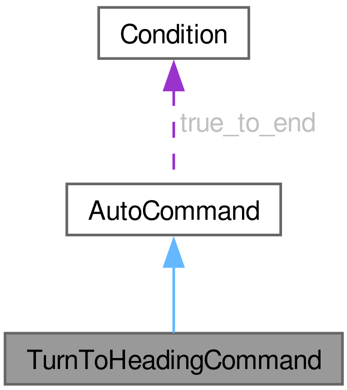
## 7.89 TurnToHeadingCommand Class Reference

`#include <drive_commands.h>`

Inheritance diagram for TurnToHeadingCommand:




Collaboration 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

## 7.89.1 Detailed Description

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

## 7.89.2 Constructor & Destructor Documentation

### 7.89.2.1 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 |

## 7.89.3 Member Function Documentation

### 7.89.3.1 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.

#### 7.89.3.2 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

## 7.90 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)

### 7.90.1 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

### 7.90.2 Constructor & Destructor Documentation

#### 7.90.2.1 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. |

#### 7.90.2.2 Vector2D() [2/2]

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

Construct a vector object from a cartesian point.

**Parameters**

| | |
|---|---|
| *p* | point_t.x , point_t.y |

### 7.90.3 Member Function Documentation

#### 7.90.3.1 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.

#### 7.90.3.2 get_mag()

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

**Returns**

the magnitude of the vector

Get the magnitude of the vector

#### 7.90.3.3 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.

#### 7.90.3.4 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.

#### 7.90.3.5 normalize()

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

Changes the magnitude of the vector to 1

**Returns**

the normalized vector

Changes the magnetude of the vector to 1

#### 7.90.3.6 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

#### 7.90.3.7 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

#### 7.90.3.8 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

#### 7.90.3.9 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:

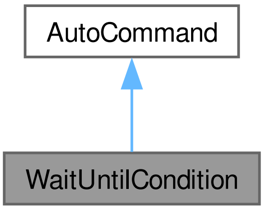
- include/utils/vector2d.h
- src/utils/vector2d.cpp

## 7.91 WaitUntilCondition Class Reference

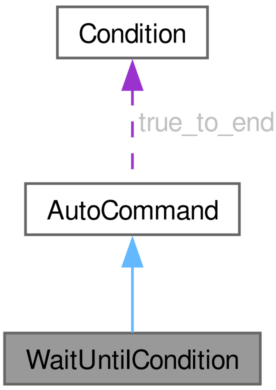
Waits until the condition is true.

`#include <auto_command.h>`

Inheritance diagram for WaitUntilCondition:




Collaboration 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

## 7.91.1 Detailed Description

Waits until the condition is true.

## 7.91.2 Constructor & Destructor Documentation

### 7.91.2.1 WaitUntilCondition()

```
WaitUntilCondition::WaitUntilCondition (
            Condition * cond )  [inline]
```

## 7.91.3 Member Function Documentation

### 7.91.3.1 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.

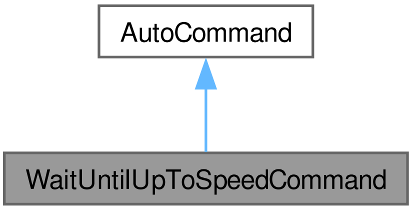
The documentation for this class was generated from the following file:

- include/utils/command_structure/auto_command.h

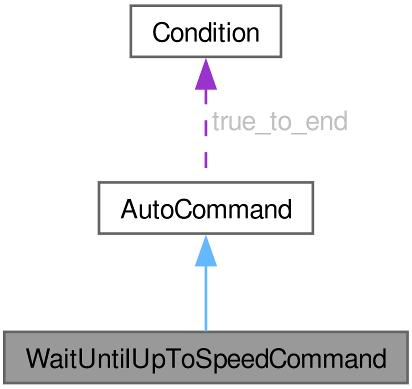
## 7.92 WaitUntilUpToSpeedCommand Class Reference

#include <flywheel_commands.h>

Inheritance diagram for WaitUntilUpToSpeedCommand:




Collaboration 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

## 7.92.1 Detailed Description

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

## 7.92.2 Constructor & Destructor Documentation

### 7.92.2.1 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 |

## 7.92.3 Member Function Documentation

### 7.92.3.1 run()

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

Run spin_manual Overrides run from AutoCommand

**Returns**

true when execution is complete, false otherwise

Reimplemented from AutoCommand.

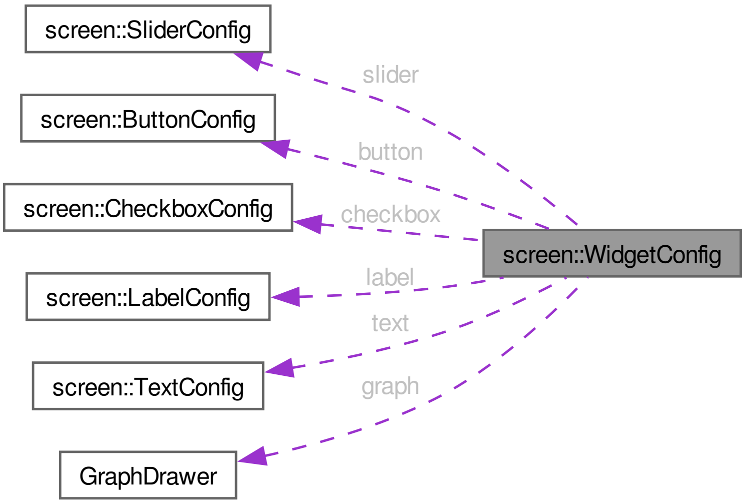
The documentation for this class was generated from the following files:

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

## 7.93 screen::WidgetConfig Struct Reference

#include <screen.h>

Collaboration diagram for screen::WidgetConfig:




**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

### 7.93.1 Member Enumeration Documentation

#### 7.93.1.1 Type

enum screen::WidgetConfig::Type

**Enumerator**

| | |
|---|---|
| Col | |
| Row | |
| Slider | |
| Button | |
| Checkbox | |
| Label | |
| Text | |
| Graph | |

## 7.93.2 Member Data Documentation

### 7.93.2.1 button

```
ButtonConfig screen::WidgetConfig::button
```

### 7.93.2.2 checkbox

```
CheckboxConfig screen::WidgetConfig::checkbox
```

### 7.93.2.3 [union]

```
union { ... } screen::WidgetConfig::config
```

### 7.93.2.4 graph

```
GraphDrawer* screen::WidgetConfig::graph
```

### 7.93.2.5 label

```
LabelConfig screen::WidgetConfig::label
```

### 7.93.2.6 slider

```
SliderConfig screen::WidgetConfig::slider
```

### 7.93.2.7 text

```
TextConfig screen::WidgetConfig::text
```

#### 7.93.2.8 type

```
Type screen::WidgetConfig::type
```

#### 7.93.2.9 widgets

```
std::vector<SizedWidget> screen::WidgetConfig::widgets
```

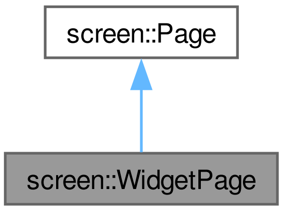
The documentation for this struct was generated from the following file:

- include/subsystems/screen.h

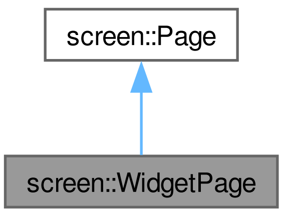
## 7.94 screen::WidgetPage Class Reference

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

Inheritance diagram for screen::WidgetPage:




Collaboration 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)*

## 7.94.1 Constructor & Destructor Documentation

### 7.94.1.1 WidgetPage()

```
screen::WidgetPage::WidgetPage (
            WidgetConfig & cfg )  [inline]
```

## 7.94.2 Member Function Documentation

### 7.94.2.1 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.

### 7.94.2.2 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

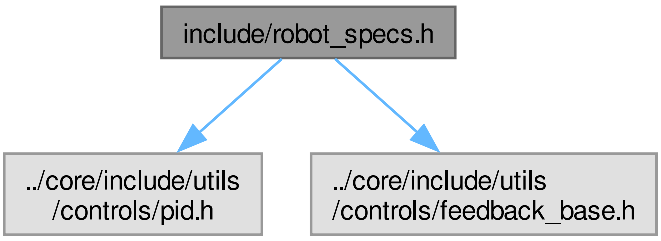
# Chapter 8

# File Documentation

## 8.1 include/robot_specs.h File Reference

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

Include dependency graph for robot_specs.h:




### Classes

- struct robot_specs_t

## 8.2 robot_specs.h

Go to the documentation of this file.

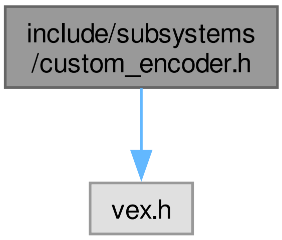
```
#pragma once
#include "../core/include/utils/controls/pid.h"
#include "../core/include/utils/controls/feedback_base.h"

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;
```

## 8.3 include/subsystems/custom_encoder.h File Reference

`#include "vex.h"`

Include dependency graph for custom_encoder.h:




### Classes

- class CustomEncoder

## 8.4 custom_encoder.h

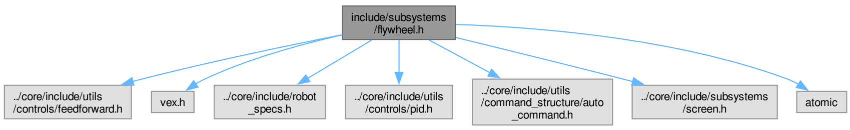
Go to the documentation of this file.

```
#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;
};
```

## 8.5 include/subsystems/flywheel.h File Reference

```
#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>
```

Include dependency graph for flywheel.h:




### Classes

- class Flywheel

## 8.6 flywheel.h

Go to the documentation of this file.

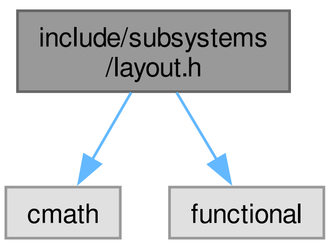
```
#pragma once

#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);
};
```

## 8.7 include/subsystems/layout.h File Reference

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

Include dependency graph for layout.h:




### Classes

- struct SliderCfg

## 8.8 layout.h

Go to the documentation of this file.

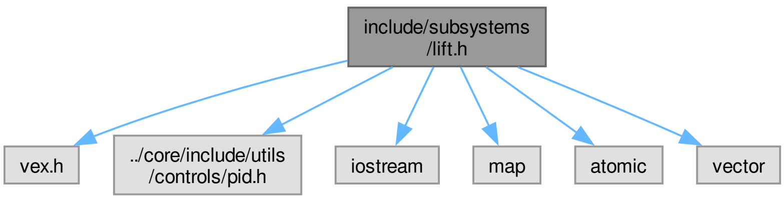
```
#include <cmath>
#include <functional>

```

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



```

## 8.9 include/subsystems/lift.h File Reference

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

Include dependency graph for lift.h:




### Classes

- class Lift< T >
- struct Lift< T >::lift_cfg_t

## 8.10 lift.h

Go to the documentation of this file.

```
#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;
```

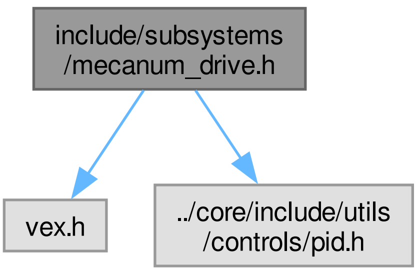
```
    if(up_btn && rev < cfg.softstop_up)
      lift_motors.spin(directionType::fwd, volt_up, voltageUnits::volt);
    else if(down_btn && rev > cfg.softstop_down && !down_hold)
      lift_motors.spin(directionType::rev, volt_down, voltageUnits::volt);
    else
      lift_motors.spin(directionType::fwd,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


};
```

## 8.11 include/subsystems/mecanum_drive.h File Reference

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

Include dependency graph for mecanum_drive.h:



### Classes

- class MecanumDrive
- struct MecanumDrive::mecanumdrive_config_t

### Macros

- #define PI 3.141592654

## 8.11.1 Macro Definition Documentation

### 8.11.1.1 PI

```
#define PI 3.141592654
```

# 8.12 mecanum_drive.h

Go to the documentation of this file.

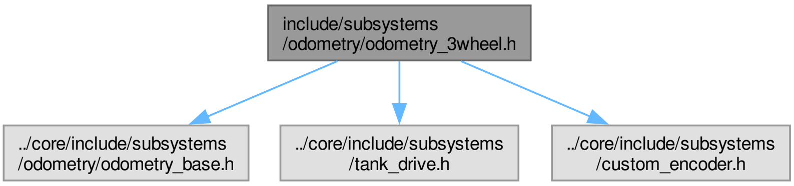
```
#pragma once

#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;

};
```

## 8.13 include/subsystems/odometry/odometry_3wheel.h File Reference

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

Include dependency graph for odometry_3wheel.h:




### Classes

- class Odometry3Wheel
- struct Odometry3Wheel::odometry3wheel_cfg_t

## 8.14 odometry_3wheel.h

Go to the documentation of this file.

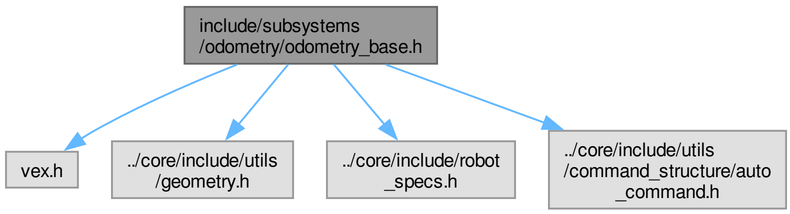
```
#pragma once
#include "../core/include/subsystems/odometry/odometry_base.h"
#include "../core/include/subsystems/tank_drive.h"
#include "../core/include/subsystems/custom_encoder.h"

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);

    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;


};
```

## 8.15 include/subsystems/odometry/odometry_base.h File Reference

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

Include dependency graph for odometry_base.h:




#### Classes

- class OdometryBase

#### Macros

- #define PI 3.141592654

### 8.15.1 Macro Definition Documentation

#### 8.15.1.1 PI

```
#define PI 3.141592654
```

## 8.16 odometry_base.h

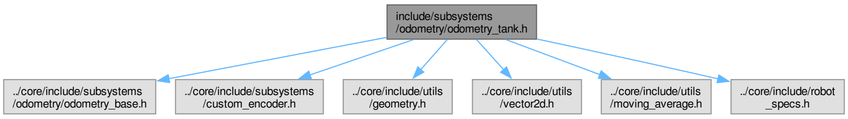
Go to the documentation of this file.

```
#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;
};
```

## 8.17 include/subsystems/odometry/odometry_tank.h File Reference

```
#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"
```

Include dependency graph for odometry_tank.h:




### Classes

- class OdometryTank

## 8.18 odometry_tank.h

Go to the documentation of this file.

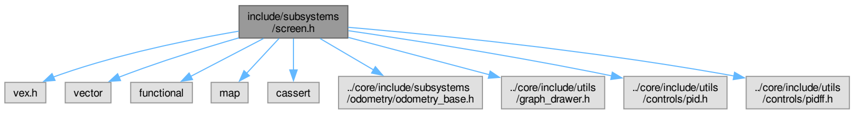
```
#pragma once

#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);

};
```

## 8.19 include/subsystems/screen.h File Reference

```
#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"
```

Include dependency graph for screen.h:




### Classes

- class screen::ButtonWidget

  *Widget that does something when you tap it. The function is only called once when you first tap it.*
- class screen::SliderWidget

  *Widget that updates a double value. Updates by reference so watch out for race conditions cuz the screen stuff lives on another thread.*
- struct screen::SliderConfig
- struct screen::ButtonConfig
- struct screen::CheckboxConfig
- struct screen::LabelConfig
- struct screen::TextConfig
- struct screen::SizedWidget
- struct screen::WidgetConfig
- class screen::Page

  *Page describes one part of the screen slideshow.*
- struct screen::ScreenRect
- class screen::WidgetPage
- class screen::StatsPage

  *Draws motor stats and battery stats to the screen.*
- class screen::OdometryPage

  *a page that shows odometry position and rotation and a map (if an sd card with the file is on)*
- class screen::FunctionPage

  *Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything.*
- class screen::PIDPage

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

### Namespaces

- namespace screen

### Typedefs

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

  *type of function needed for update*
- using screen::draw_func_t = std::function<void(vex::brain::lcd &screen, bool, unsigned int)>

  *type of function needed for draw*

### Functions

- void screen::draw_widget (WidgetConfig &widget, ScreenRect rect)
- void screen::start_screen (vex::brain::lcd &screen, std::vector< Page * > pages, int first_page=0)

    *Start the screen background task. Once you start this, no need to draw to the screen manually elsewhere.*
- void screen::next_page ()
- void screen::prev_page ()
- void screen::stop_screen ()

    *stops the screen. If you have a drive team that hates fun call this at the start of opcontrol*

## 8.20 screen.h

Go to the documentation of this file.

```
#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
```

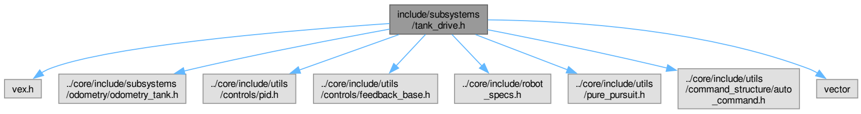
```
{
    std::function<void(bool)> onupdate;
};
struct LabelConfig
{
    std::string label;
};

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;
    };

}
```

## 8.21 include/subsystems/tank_drive.h File Reference

```
#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>
```

Include dependency graph for tank_drive.h:




### Classes

- class TankDrive

### Macros

- #define PI 3.141592654

### 8.21.1 Macro Definition Documentation

#### 8.21.1.1 PI

```
#define PI 3.141592654
```

## 8.22 tank_drive.h

Go to the documentation of this file.

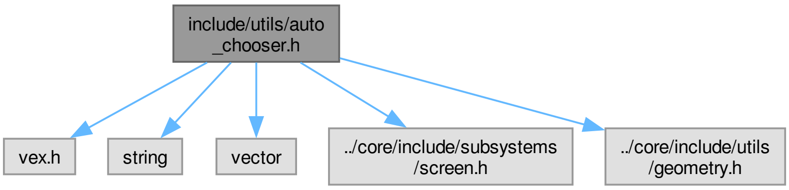
```
#pragma once

#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;
};
```

## 8.23 include/utils/auto_chooser.h File Reference

```
#include "vex.h"
#include <string>
#include <vector>
#include "../core/include/subsystems/screen.h"
#include "../core/include/utils/geometry.h"
```

Include dependency graph for auto_chooser.h:




### Classes

- class AutoChooser
- struct AutoChooser::entry_t

## 8.24 auto_chooser.h

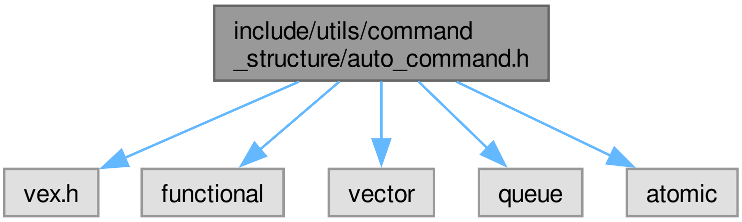
Go to the documentation of this file.

```
#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 ;
};
```

## 8.25 include/utils/command_structure/auto_command.h File Reference

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

Include dependency graph for auto_command.h:




### Classes

- class Condition
- class AutoCommand
- class FunctionCommand
- class TimesTestedCondition
- class FunctionCondition

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

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

    *Waits until the condition is true.*
- class InOrder

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

    *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.*
- class Branch

    *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.*
- class Async

    *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.*
- class RepeatUntil

## 8.26 auto_command.h

Go to the documentation of this file.

```
#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
```

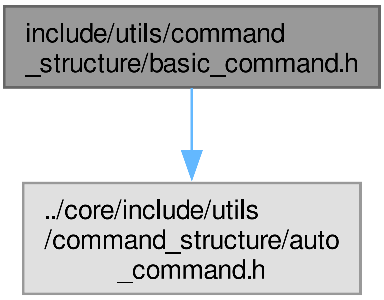
```
{
public:
  FunctionCondition(
      std::function<bool()> cond, std::function<void(void)> timeout = []() {}) : cond(cond), timeout(timeout)
  {
  }
  bool test() override;

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;
};
```

## 8.27 include/utils/command_structure/basic_command.h File Reference

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

Include dependency graph for basic_command.h:




### Classes

- class BasicSpinCommand
- class BasicStopCommand
- class BasicSolenoidSet

## 8.28 basic_command.h

Go to the documentation of this file.

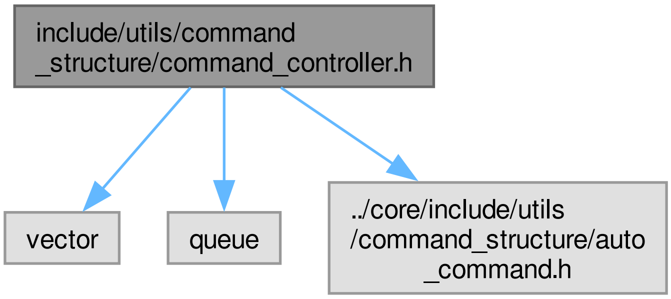
```

#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;
};
```

## 8.29 include/utils/command_structure/command_controller.h File Reference

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

Include dependency graph for command_controller.h:




## Classes

- class CommandController

# 8.30 command_controller.h

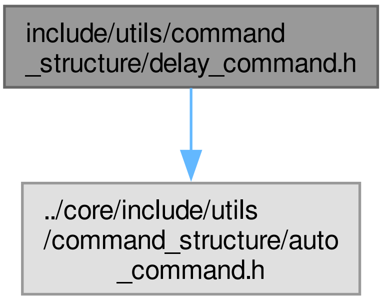
Go to the documentation of this file.

```

#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; };
};
```

## 8.31 include/utils/command_structure/delay_command.h File Reference

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

Include dependency graph for delay_command.h:




### Classes

- class DelayCommand

## 8.32 delay_command.h

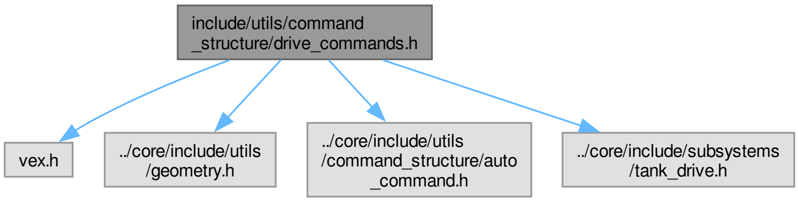
Go to the documentation of this file.

```
#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;
};
```

## 8.33 include/utils/command_structure/drive_commands.h File Reference

```
#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"`
Include dependency graph for drive_commands.h:




### Classes

- class DriveForwardCommand
- class TurnDegreesCommand
- class DriveToPointCommand
- class TurnToHeadingCommand
- class PurePursuitCommand
- class DriveStopCommand
- class OdomSetPosition

## 8.34 drive_commands.h

Go to the documentation of this file.

```

#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:
```

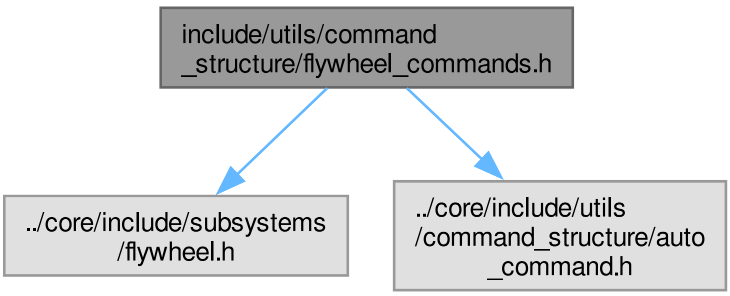
```
    TurnDegreesCommand(TankDrive &drive_sys, Feedback &feedback, double degrees, double max_speed = 1, double end_speed = 0);
    bool run() override;
    void on_timeout() override;
  private:
    // drive system to run the function on
    TankDrive &drive_sys;
    // feedback controller to use
    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;
};
```

## 8.35 include/utils/command_structure/flywheel_commands.h File Reference

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

Include dependency graph for flywheel_commands.h:




### Classes

- class SpinRPMCommand
- class WaitUntilUpToSpeedCommand
- class FlywheelStopCommand
- class FlywheelStopMotorsCommand
- class FlywheelStopNonTasksCommand

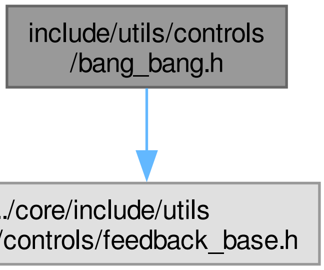
## 8.36 flywheel_commands.h

Go to the documentation of this file.

```
#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;
};
```

## 8.37 include/utils/controls/bang_bang.h File Reference

`#include "../core/include/utils/controls/feedback_base.h"`
Include dependency graph for bang_bang.h:




### Classes

- class BangBang

## 8.38 bang_bang.h

Go to the documentation of this file.

```
#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;
};
```

## 8.39 include/utils/controls/feedback_base.h File Reference

### Classes

- class Feedback

## 8.40 feedback_base.h

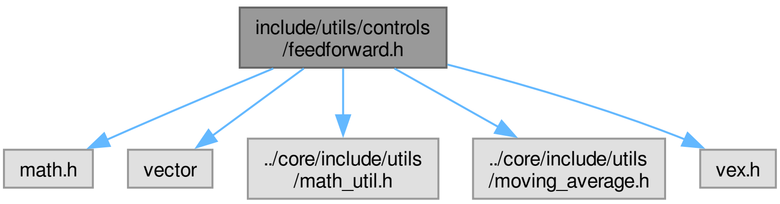
Go to the documentation of this file.

```
#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;


};
```

## 8.41 include/utils/controls/feedforward.h File Reference

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

Include dependency graph for feedforward.h:




### Classes

- class FeedForward
- struct FeedForward::ff_config_t

### Functions

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

### 8.41.1 Function Documentation

#### 8.41.1.1 tune_feedforward()

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

tune_feedforward takes a group of motors and finds the feedforward conifg parameters automagically.

**Parameters**

| | |
|---|---|
| *motor* | the motor group to use |
| *pct* | Maximum velocity in percent (0->1.0) |
| *duration* | Amount of time the motors spin for the test |

**Returns**

A tuned feedforward object

## 8.42 feedforward.h

Go to the documentation of this file.

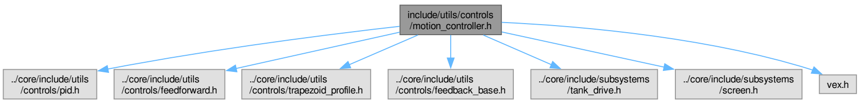
```
#pragma once

#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);
```

## 8.43 include/utils/controls/motion_controller.h File Reference

```
#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"`

Include dependency graph for motion_controller.h:




### Classes

- class MotionController
- struct MotionController::m_profile_cfg_t

## 8.44 motion_controller.h

Go to the documentation of this file.

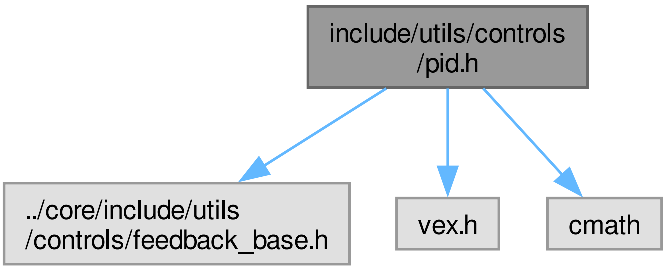
```
#pragma once
#include "../core/include/utils/controls/pid.h"
#include "../core/include/utils/controls/feedforward.h"
#include "../core/include/utils/controls/trapezoid_profile.h"
#include "../core/include/utils/controls/feedback_base.h"
#include "../core/include/subsystems/tank_drive.h"
#include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

};
```

## 8.45 include/utils/controls/pid.h File Reference

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

Include dependency graph for pid.h:




### Classes

- class PID
- struct PID::pid_config_t

## 8.46 pid.h

Go to the documentation of this file.

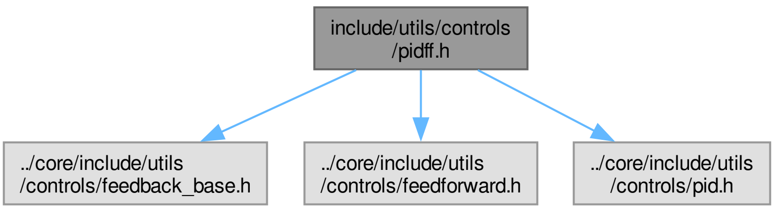
```
#pragma once

#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;
};
```

## 8.47 include/utils/controls/pidff.h File Reference

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

Include dependency graph for pidff.h:



### Classes

- class PIDFF

## 8.48 pidff.h

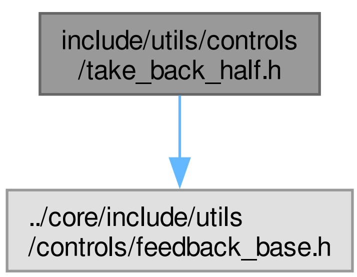
Go to the documentation of this file.

```
#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;
};
```

## 8.49 include/utils/controls/take_back_half.h File Reference

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

Include dependency graph for take_back_half.h:



### Classes

- class TakeBackHalf

  *A velocity controller.*

## 8.50 take_back_half.h

Go to the documentation of this file.

```
#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;
};
```

## 8.51 include/utils/controls/trapezoid_profile.h File Reference

### Classes

- struct motion_t
- struct trapezoid_profile_segment_t
- class TrapezoidProfile

### Variables

- const int MAX_TRAPEZOID_PROFILE_SEGMENTS = 4

### 8.51.1 Variable Documentation

#### 8.51.1.1 MAX_TRAPEZOID_PROFILE_SEGMENTS

```
const int MAX_TRAPEZOID_PROFILE_SEGMENTS = 4
```

## 8.52 trapezoid_profile.h

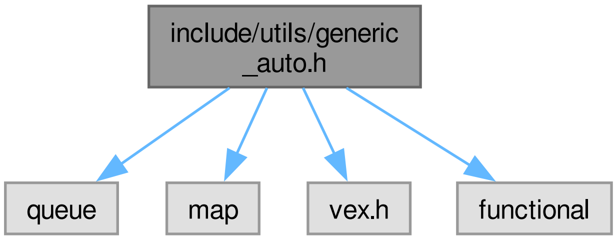
Go to the documentation of this file.

```
#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);
};
```

## 8.53 include/utils/generic_auto.h File Reference

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

Include dependency graph for generic_auto.h:




### Classes

- class GenericAuto

### Typedefs

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

## 8.53.1 Typedef Documentation

### 8.53.1.1 state_ptr

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

# 8.54 generic_auto.h

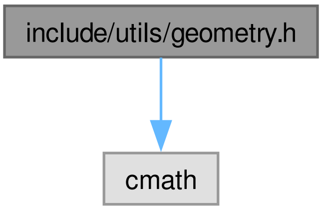
Go to the documentation of this file.

```
#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;

};
```

## 8.55 include/utils/geometry.h File Reference

`#include <cmath>`

Include dependency graph for geometry.h:




### Classes

- struct point_t
- struct pose_t
- struct Rect
- struct Mat2

## 8.56 geometry.h

Go to the documentation of this file.

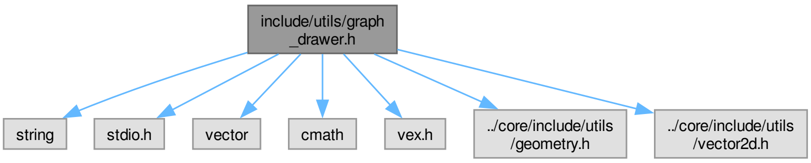
```
#pragma once
#include <cmath>

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};
    }
};
```

## 8.57 include/utils/graph_drawer.h File Reference

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

Include dependency graph for graph_drawer.h:




### Classes

- class GraphDrawer

## 8.58 graph_drawer.h

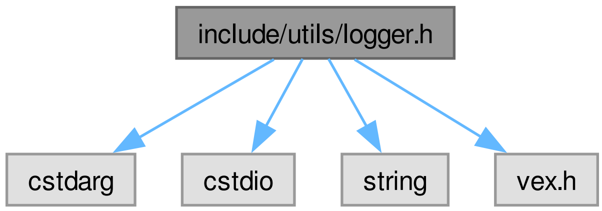
Go to the documentation of this file.

```
#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;
};
```

## 8.59 include/utils/logger.h File Reference

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

Include dependency graph for logger.h:




#### Classes

- class Logger

  *Class to simplify writing to files.*

#### Enumerations

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

  *possible values for log filtering*

### 8.59.1 Enumeration Type Documentation

#### 8.59.1.1 LogLevel

```
enum LogLevel
```

possible values for log filtering

**Enumerator**

| | |
|---|---|
| DEBUG | |
| NOTICE | |
| WARNING | |
| ERROR | |
| CRITICAL | |
| TIME | |

## 8.60 logger.h

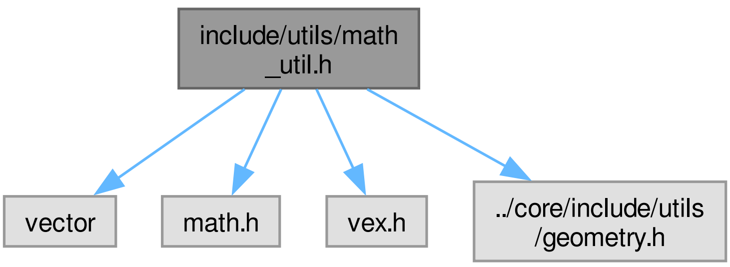
Go to the documentation of this file.

```
#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, ...);
};
```

## 8.61 include/utils/math_util.h File Reference

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

Include dependency graph for math_util.h:



### Functions

- double clamp (double value, double low, double high)
- double lerp (double a, double b, double t)

  *Linearly intERPolate between values.*
- double sign (double x)
- double wrap_angle_deg (double input)
- double wrap_angle_rad (double input)
- double variance (std::vector< double > const &values, double mean)
- double mean (std::vector< double > const &values)
- double covariance (std::vector< std::pair< double, double > > const &points, double meanx, double meany)
- 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)

## 8.61.1 Function Documentation

### 8.61.1.1 calculate_linear_regression()

```
std::pair< double, double > calculate_linear_regression (
            std::vector< std::pair< double, double > > const & points )
```

### 8.61.1.2 clamp()

```
double clamp (
            double val,
            double low,
            double high )
```

Constrain the input between a minimum and a maximum value

**Parameters**

| | |
|---|---|
| *val* | the value to be restrained |
| *low* | the minimum value that will be returned |
| *high* | the maximum value that will be returned |

### 8.61.1.3 covariance()

```
double covariance (
            std::vector< std::pair< double, double > > const & points,
            double meanx,
            double meany )
```

### 8.61.1.4 estimate_path_length()

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

#### 8.61.1.5 lerp()

```
double lerp (
            double a,
            double b,
            double t )
```

Linearly intERPolate between values.

**Parameters**

| | |
|---|---|
| *a* | at t = 0, output = a |
| *b* | at t = 1, output = b |

**Returns**

a linear mixing of a and b according to t

#### 8.61.1.6 mean()

```
double mean (
            std::vector< double > const & values )
```

#### 8.61.1.7 sign()

```
double sign (
            double x )
```

Returns the sign of a number

**Parameters**

| | |
|---|---|
| *x* | |

returns the sign +/-1 of x. 0 if x is 0

Returns the sign of a number

**Parameters**

| | |
|---|---|
| *x* | |

returns the sign +/-1 of x. special case at 0 it returns +1

#### 8.61.1.8 variance()

```
double variance (
            std::vector< double > const & values,
            double mean )
```

#### 8.61.1.9 wrap_angle_deg()

```
double wrap_angle_deg (
            double input )
```

#### 8.61.1.10 wrap_angle_rad()

```
double wrap_angle_rad (
            double input )
```

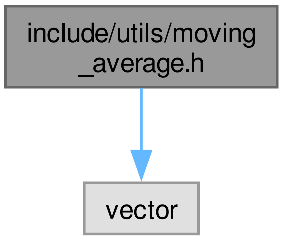
## 8.62 math_util.h

Go to the documentation of this file.

```
#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);
```

## 8.63 include/utils/moving_average.h File Reference

#include <vector>
Include dependency graph for moving_average.h:




### Classes

- class Filter
- class MovingAverage
- class ExponentialMovingAverage

## 8.64 moving_average.h

Go to the documentation of this file.

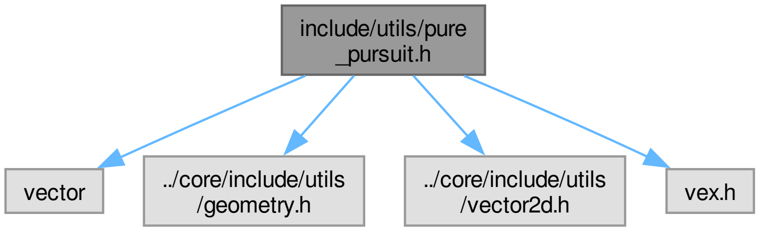
```
#pragma once
#include <vector>

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
};
```

## 8.65 include/utils/pure_pursuit.h File Reference

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

Include dependency graph for pure_pursuit.h:




### Classes

- class PurePursuit::Path
- struct PurePursuit::spline
- struct PurePursuit::hermite_point

### Namespaces

- namespace PurePursuit

### Functions

- std::vector< point_t > PurePursuit::line_circle_intersections (point_t center, double r, point_t point1, point_t point2)
- point_t PurePursuit::get_lookahead (const std::vector< point_t > &path, pose_t robot_loc, double radius)
- std::vector< point_t > PurePursuit::inject_path (const std::vector< point_t > &path, double spacing)
- std::vector< point_t > PurePursuit::smooth_path (const std::vector< point_t > &path, double weight_data, double weight_smooth, double tolerance)
- std::vector< point_t > PurePursuit::smooth_path_cubic (const std::vector< point_t > &path, double res)
- std::vector< point_t > PurePursuit::smooth_path_hermite (const std::vector< hermite_point > &path, double step)
- double PurePursuit::estimate_remaining_dist (const std::vector< point_t > &path, pose_t robot_pose, double radius)

## 8.66 pure_pursuit.h

Go to the documentation of this file.

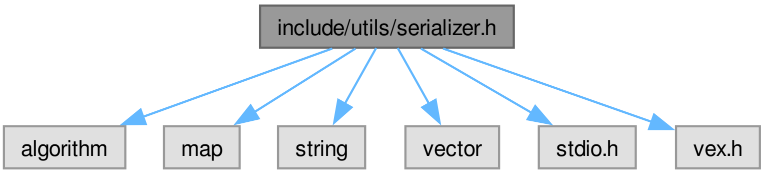
```
#pragma once

#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);

}
```

## 8.67 include/utils/serializer.h File Reference

```
#include <algorithm>
#include <map>
#include <string>
#include <vector>
#include <stdio.h>
#include <vex.h>
```

Include dependency graph for serializer.h:




## Classes

- class Serializer

    *Serializes Arbitrary data to a file on the SD Card.*

## Variables

- const char serialization_separator = '$'

    *character that will be used to seperate values*
- const std::size_t MAX_FILE_SIZE = 4096

    *max file size that the system can deal with*

## 8.67.1 Variable Documentation

### 8.67.1.1 MAX_FILE_SIZE

```
const std::size_t MAX_FILE_SIZE = 4096
```

max file size that the system can deal with

### 8.67.1.2 serialization_separator

```
const char serialization_separator = '$'
```

character that will be used to seperate values

## 8.68 serializer.h

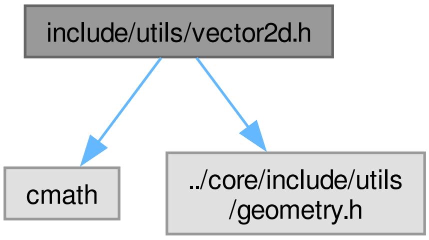
Go to the documentation of this file.

```
#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);
};
```

## 8.69 include/utils/vector2d.h File Reference

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

Include dependency graph for vector2d.h:




### Classes

- class Vector2D

### Macros

- #define PI 3.141592654

### Functions

- double deg2rad (double deg)
- double rad2deg (double r)

## 8.69.1 Macro Definition Documentation

### 8.69.1.1 PI

```
#define PI 3.141592654
```

## 8.69.2 Function Documentation

### 8.69.2.1 deg2rad()

```
double deg2rad (
            double deg )
```

General function for converting degrees to radians

**Parameters**

| | |
|---|---|
| *deg* | the angle in degrees |

**Returns**

the angle in radians

General function for converting degrees to radians

#### 8.69.2.2 rad2deg()

```
double rad2deg (
            double rad )
```

General function for converting radians to degrees

**Parameters**

| | |
|---|---|
| *r* | the angle in radians |

**Returns**

the angle in degrees

General function for converting radians to degrees

## 8.70 vector2d.h

Go to the documentation of this file.

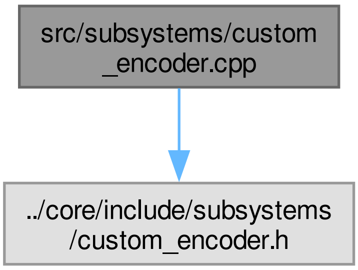
```
#pragma once


#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);
```

## 8.71 README.md File Reference

## 8.72 src/subsystems/custom_encoder.cpp File Reference

```
#include "../core/include/subsystems/custom_encoder.h"
```

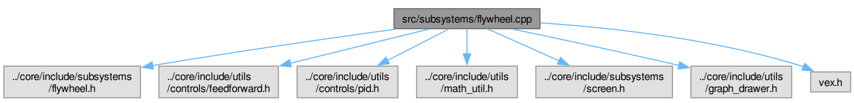
Include dependency graph for custom_encoder.cpp:




## 8.73 src/subsystems/flywheel.cpp File Reference

```
#include "../core/include/subsystems/flywheel.h"
#include "../core/include/utils/controls/feedforward.h"
#include "../core/include/utils/controls/pid.h"
#include "../core/include/utils/math_util.h"
#include "../core/include/subsystems/screen.h"
#include "../core/include/utils/graph_drawer.h"
#include "vex.h"
```

Include dependency graph for flywheel.cpp:




### Classes

- class FlywheelPage

### Functions

- int spinRPMTask (void ∗wheelPointer)

### 8.73.1 Function Documentation

#### 8.73.1.1 spinRPMTask()

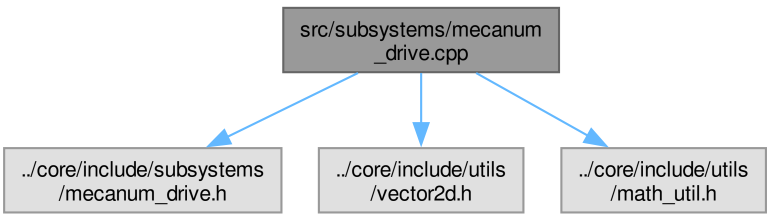
```
int spinRPMTask (
            void * wheelPointer )
```

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

## 8.74 src/subsystems/mecanum_drive.cpp File Reference

```
#include "../core/include/subsystems/mecanum_drive.h"
#include "../core/include/utils/vector2d.h"
#include "../core/include/utils/math_util.h"
```

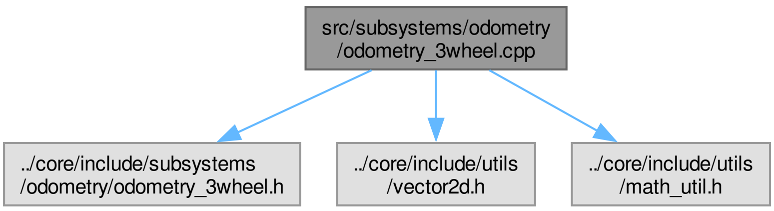
Include dependency graph for mecanum_drive.cpp:




## 8.75 src/subsystems/odometry/odometry_3wheel.cpp File Reference

```
#include "../core/include/subsystems/odometry/odometry_3wheel.h"
#include "../core/include/utils/vector2d.h"
#include "../core/include/utils/math_util.h"
```

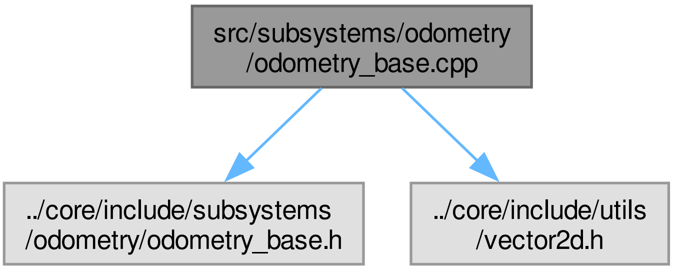
Include dependency graph for odometry_3wheel.cpp:



## 8.76 src/subsystems/odometry/odometry_base.cpp File Reference

```
#include "../core/include/subsystems/odometry/odometry_base.h"
#include "../core/include/utils/vector2d.h"
```

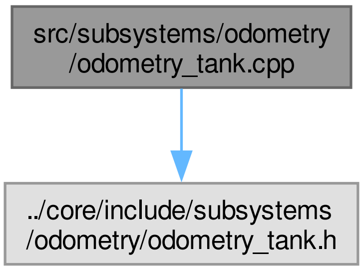
Include dependency graph for odometry_base.cpp:




## 8.77 src/subsystems/odometry/odometry_tank.cpp File Reference

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

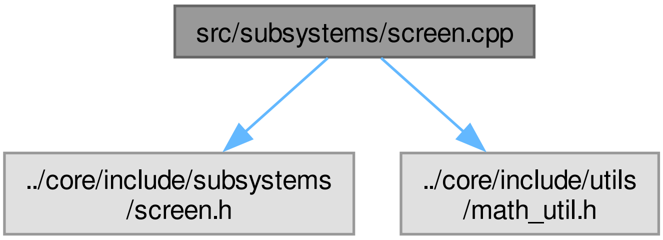
Include dependency graph for odometry_tank.cpp:




## 8.78 src/subsystems/screen.cpp File Reference

```
#include "../core/include/subsystems/screen.h"
#include "../core/include/utils/math_util.h"
```

Include dependency graph for screen.cpp:




### Classes

- struct screen::ScreenData

  *The ScreenData class holds the data that will be passed to the screen thread you probably shouldnt have to use it.*

### Namespaces

- namespace screen

### Functions

- void screen::draw_label (vex::brain::lcd &scr, std::string lbl, ScreenRect rect)
- void screen::draw_widget (vex::brain::lcd &scr, WidgetConfig &widget, ScreenRect rect)
- void screen::start_screen (vex::brain::lcd &screen, std::vector< Page * > pages, int first_page=0)

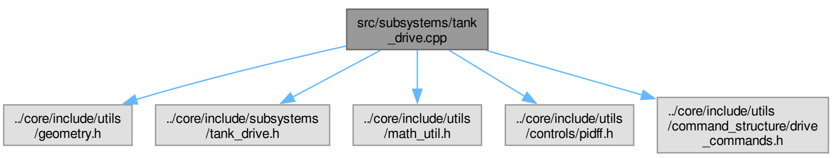
  *Start the screen background task. Once you start this, no need to draw to the screen manually elsewhere.*
- void screen::stop_screen ()

  *stops the screen. If you have a drive team that hates fun call this at the start of opcontrol*
- void screen::prev_page ()
- void screen::next_page ()
- int screen::in_to_px (double in)

## 8.79 src/subsystems/tank_drive.cpp File Reference

```
#include "../core/include/utils/geometry.h"
#include "../core/include/subsystems/tank_drive.h"
#include "../core/include/utils/math_util.h"
#include "../core/include/utils/controls/pidff.h"
#include "../core/include/utils/command_structure/drive_commands.h"
```

Include dependency graph for tank_drive.cpp:



**Variables**

- bool captured_position = false
- bool was_breaking = false

### 8.79.1 Variable Documentation

#### 8.79.1.1 captured_position

```
bool captured_position = false
```

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

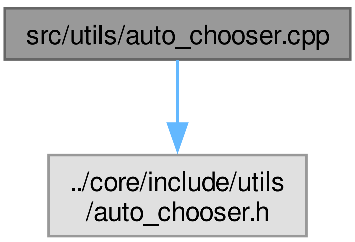
#### 8.79.1.2 was_breaking

```
bool was_breaking = false
```

## 8.80 src/utils/auto_chooser.cpp File Reference

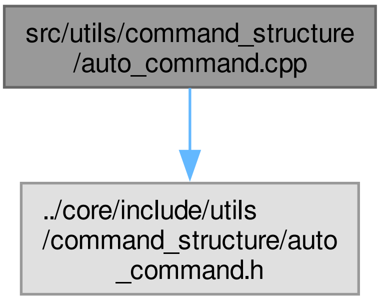
```
#include "../core/include/utils/auto_chooser.h"
```

Include dependency graph for auto_chooser.cpp:



## 8.81 src/utils/command_structure/auto_command.cpp File Reference

`#include "../core/include/utils/command_structure/auto_command.h"`
Include dependency graph for auto_command.cpp:

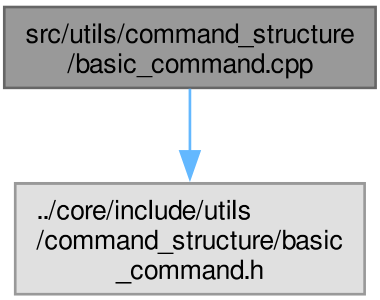



### Classes

- class OrCondition
- class AndCondition
- struct parallel_runner_info

## 8.82 src/utils/command_structure/basic_command.cpp File Reference

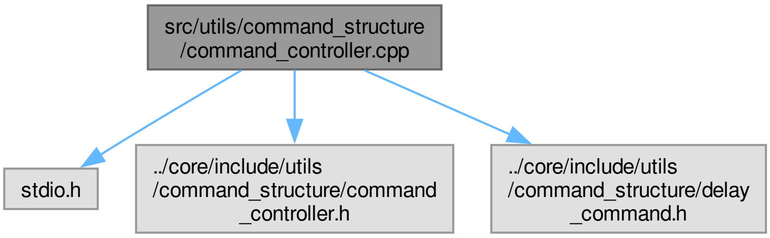
`#include "../core/include/utils/command_structure/basic_command.h"`
Include dependency graph for basic_command.cpp:



## 8.83 src/utils/command_structure/command_controller.cpp File Reference

```
#include <stdio.h>
#include "../core/include/utils/command_structure/command_controller.h"
#include "../core/include/utils/command_structure/delay_command.h"
```

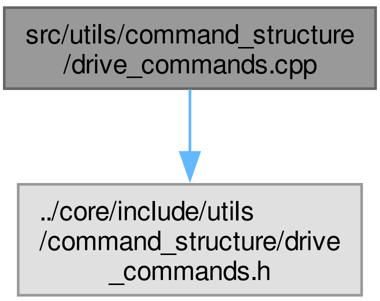
Include dependency graph for command_controller.cpp:




## 8.84 src/utils/command_structure/drive_commands.cpp File Reference

```
#include "../core/include/utils/command_structure/drive_commands.h"
```

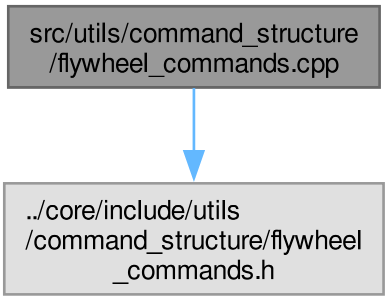
Include dependency graph for drive_commands.cpp:




## 8.85 src/utils/command_structure/flywheel_commands.cpp File Reference

```
#include "../core/include/utils/command_structure/flywheel_commands.h"
```

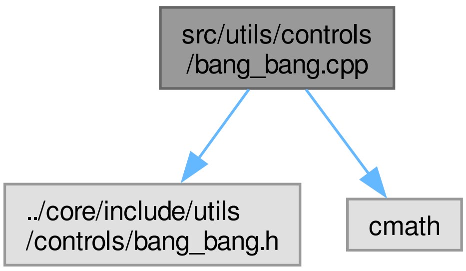
Include dependency graph for flywheel_commands.cpp:




## 8.86 src/utils/controls/bang_bang.cpp File Reference

```
#include "../core/include/utils/controls/bang_bang.h"
#include <cmath>
```

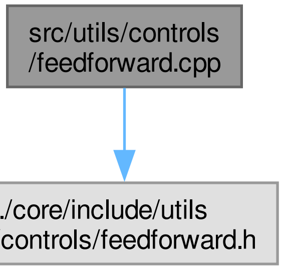
Include dependency graph for bang_bang.cpp:




## 8.87 src/utils/controls/feedforward.cpp File Reference

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

Include dependency graph for feedforward.cpp:




### Functions

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

## 8.87.1 Function Documentation

### 8.87.1.1 tune_feedforward()

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

tune_feedforward takes a group of motors and finds the feedforward conifg parameters automagically.

**Parameters**

| | |
|---|---|
| *motor* | the motor group to use |
| *pct* | Maximum velocity in percent (0->1.0) |
| *duration* | Amount of time the motors spin for the test |

**Returns**

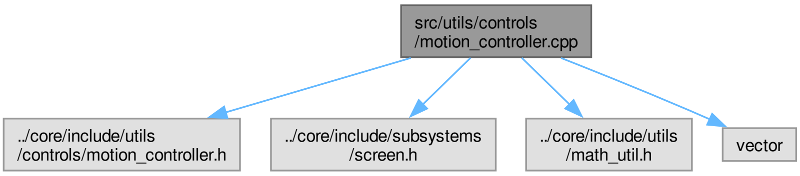
A tuned feedforward object

## 8.88 src/utils/controls/motion_controller.cpp File Reference

```
#include "../core/include/utils/controls/motion_controller.h"
#include "../core/include/subsystems/screen.h"
#include "../core/include/utils/math_util.h"
```

```
#include <vector>
```

Include dependency graph for motion_controller.cpp:



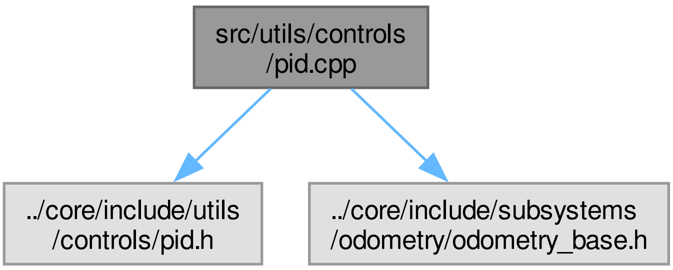

### Classes

- class MotionControllerPage

## 8.89 src/utils/controls/pid.cpp File Reference

```
#include "../core/include/utils/controls/pid.h"
#include "../core/include/subsystems/odometry/odometry_base.h"
```

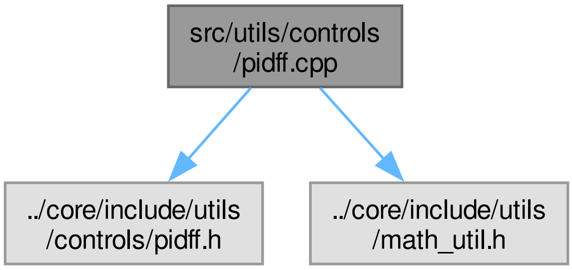
Include dependency graph for pid.cpp:




## 8.90 src/utils/controls/pidff.cpp File Reference

```
#include "../core/include/utils/controls/pidff.h"
#include "../core/include/utils/math_util.h"
```

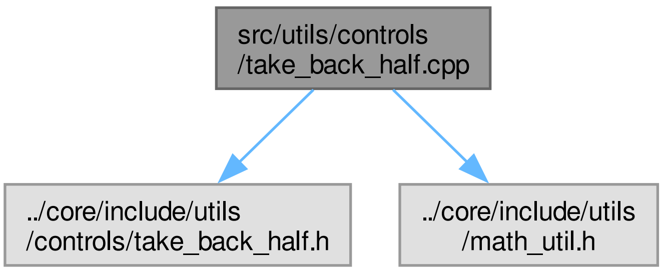
Include dependency graph for pidff.cpp:




## 8.91 src/utils/controls/take_back_half.cpp File Reference

```
#include "../core/include/utils/controls/take_back_half.h"
#include "../core/include/utils/math_util.h"
```

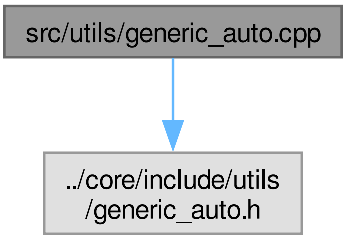
Include dependency graph for take_back_half.cpp:




## 8.92 src/utils/generic_auto.cpp File Reference

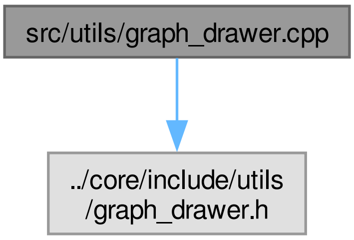
```
#include "../core/include/utils/generic_auto.h"
```

Include dependency graph for generic_auto.cpp:




## 8.93 src/utils/graph_drawer.cpp File Reference

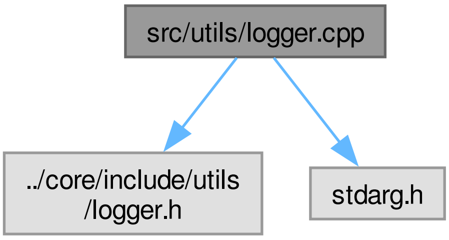
```
#include "../core/include/utils/graph_drawer.h"
```
Include dependency graph for graph_drawer.cpp:




## 8.94 src/utils/logger.cpp File Reference

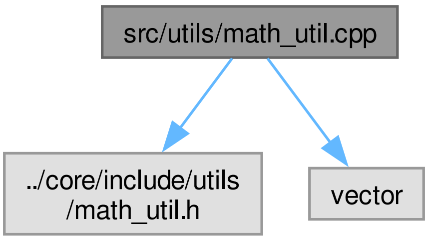
```
#include "../core/include/utils/logger.h"
#include <stdarg.h>
```

Include dependency graph for logger.cpp:




## 8.95 src/utils/math_util.cpp File Reference

```
#include "../core/include/utils/math_util.h"
#include <vector>
```
Include dependency graph for math_util.cpp:




### Macros

- #define PI 3.141592654

### Functions

- double clamp (double val, double low, double high)
- double lerp (double a, double b, double t)

    *Linearly intERPolate between values.*
- double sign (double x)
- double wrap_angle_deg (double input)
- double wrap_angle_rad (double input)
- double mean (std::vector< double > const &values)

- double variance (std::vector< double > const &values, double mean)
- double covariance (std::vector< std::pair< double, double > > const &points, double meanx, double meany)
- 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)

## 8.95.1 Macro Definition Documentation

### 8.95.1.1 PI

```
#define PI 3.141592654
```

## 8.95.2 Function Documentation

### 8.95.2.1 calculate_linear_regression()

```
std::pair< double, double > calculate_linear_regression (
            std::vector< std::pair< double, double > > const & points )
```

### 8.95.2.2 clamp()

```
double clamp (
            double val,
            double low,
            double high )
```

Constrain the input between a minimum and a maximum value

**Parameters**

| | |
|---|---|
| *val* | the value to be restrained |
| *low* | the minimum value that will be returned |
| *high* | the maximum value that will be returned |

### 8.95.2.3 covariance()

```
double covariance (
            std::vector< std::pair< double, double > > const & points,
            double meanx,
            double meany )
```

### 8.95.2.4 estimate_path_length()

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

#### 8.95.2.5 lerp()

```
double lerp (
            double a,
            double b,
            double t )
```

Linearly intERPolate between values.

**Parameters**

| | |
|---|---|
| *a* | at t = 0, output = a |
| *b* | at t = 1, output = b |

**Returns**

a linear mixing of a and b according to t

#### 8.95.2.6 mean()

```
double mean (
            std::vector< double > const & values )
```

#### 8.95.2.7 sign()

```
double sign (
            double x )
```

Returns the sign of a number

**Parameters**

| | |
|---|---|
| *x* | |

returns the sign +/-1 of x. special case at 0 it returns +1

#### 8.95.2.8 variance()

```
double variance (
            std::vector< double > const & values,
            double mean )
```

#### 8.95.2.9 wrap_angle_deg()

```
double wrap_angle_deg (
            double input )
```

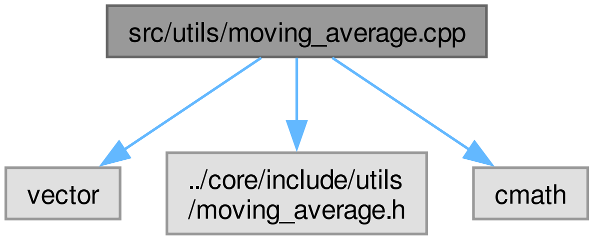
#### 8.95.2.10 wrap_angle_rad()

```
double wrap_angle_rad (
            double input )
```

## 8.96 src/utils/moving_average.cpp File Reference

```
#include <vector>
#include "../core/include/utils/moving_average.h"
#include <cmath>
```

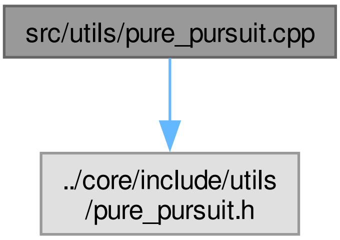
Include dependency graph for moving_average.cpp:




## 8.97 src/utils/pure_pursuit.cpp File Reference

```
#include "../core/include/utils/pure_pursuit.h"
```

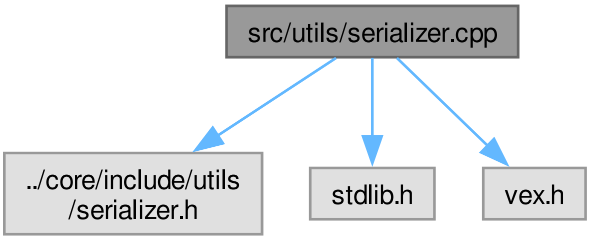
Include dependency graph for pure_pursuit.cpp:



## 8.98 src/utils/serializer.cpp File Reference

```
#include "../core/include/utils/serializer.h"
#include "stdlib.h"
#include "vex.h"
```

Include dependency graph for serializer.cpp:




### Functions

- template< typename T >
  std::vector< char > to_bytes (T value)

  *Convert type to bytes. Overload this for non integer types.*
- template<> std::vector< char > to_bytes< std::string > (std::string str)
- template< typename T >
  T from_bytes (std::vector< char >::const_iterator &position)

  *Convert bytes to a type.*
- template<> std::string from_bytes (std::vector< char >::const_iterator &position)
- std::string sanitize_name (std::string s)

  *Replaces funny characters in names so they don't mess with serialization specifiers.*

### 8.98.1 Function Documentation

#### 8.98.1.1 from_bytes() [1/2]

```
template<typename T >
T from_bytes (
            std::vector< char >::const_iterator & position )
```

Convert bytes to a type.

**Parameters**

| | |
|---|---|
| *gets* | data from arbitrary bytes. Overload this for non integer types |

#### 8.98.1.2 from_bytes() [2/2]

```
template<>
std::string from_bytes (
            std::vector< char >::const_iterator & position )
```

#### 8.98.1.3 sanitize_name()

```
std::string sanitize_name (
            std::string s )
```

Replaces funny characters in names so they don't mess with serialization specifiers.

#### 8.98.1.4 to_bytes()

```
template<typename T >
std::vector< char > to_bytes (
            T value )
```

Convert type to bytes. Overload this for non integer types.

**Parameters**

| | |
|---|---|
| *value* | value to convert |

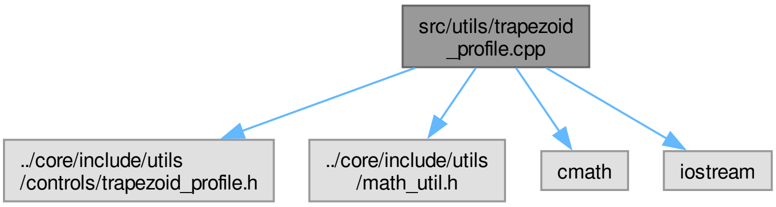
#### 8.98.1.5 to_bytes< std::string >()

```
template<>
std::vector< char > to_bytes< std::string > (
            std::string str )
```

## 8.99 src/utils/trapezoid_profile.cpp File Reference

```
#include "../core/include/utils/controls/trapezoid_profile.h"
#include "../core/include/utils/math_util.h"
#include <cmath>
#include <iostream>
```

Include dependency graph for trapezoid_profile.cpp:




### Functions

- double calc_pos (double t, double a, double v, double si)
- double calc_vel (double t, double a, double vi)

### Variables

- const double EPSILON = 0.000005

## 8.99.1 Function Documentation

### 8.99.1.1 calc_pos()

```
double calc_pos (
            double t,
            double a,
            double v,
            double si )  [inline]
```

### 8.99.1.2 calc_vel()

```
double calc_vel (
            double t,
            double a,
            double vi )  [inline]
```

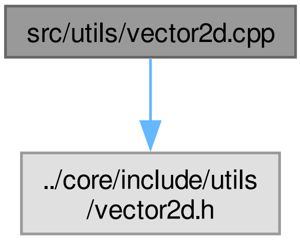
## 8.99.2 Variable Documentation

### 8.99.2.1 EPSILON

```
const double EPSILON = 0.000005
```

## 8.100 src/utils/vector2d.cpp File Reference

```
#include "../core/include/utils/vector2d.h"
```
Include dependency graph for vector2d.cpp:




**Functions**

- double deg2rad (double deg)
- double rad2deg (double rad)

### 8.100.1 Function Documentation

#### 8.100.1.1 deg2rad()

```
double deg2rad (
            double deg )
```

General function for converting degrees to radians

#### 8.100.1.2 rad2deg()

```
double rad2deg (
            double rad )
```

General function for converting radians to degrees

# Index