RIT VEXU Core API

Generated by Doxygen 1.13.2

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

If you just want to start a project with Core, make a fork of the `Fork Template` and follow it's instructions.

To setup core for an existing project:

1. Create a new vex project (using the VSCode extension or other methods)
2. Initialize a git repository for the project
3. Execute `git subtree add --prefix=core https://github.com/RIT-VEX-U/↩ Core.git main`
4. Update the vex Makefile (or any other build system) to know about the core files (`core/src` for source files, `core/include` for headers) (See `here` for an example)
5. Enable `Eigen` (Latest supported version is 3.4.0):
   - `mkdir vendor`
   - `git submodule add https://gitlab.com/libeigen/eigen.git vendor/eigen`
   - `cd vendor/eigen`
   - `git checkout 3.4.0`
   - Add the following to the `makefile` to give Core access to the library: `INC += -Ivendor/eigen` (See `here` for an example)

If you only wish to use a single version of Core, you can simply clone core/ into your project and add the core source and header files to your makefile.

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
  - Tank (Differential)
  - `N-Pod`
- 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)

2 Hierarchical Index

2.1 Class Hierarchy

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

3 Class Index

3.1 Class List

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

4 Class Documentation

4.1 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>
```

4.1.1 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.

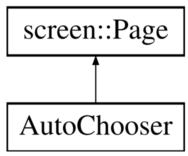
The documentation for this class was generated from the following files:

- auto_command.h
- auto_command.cpp

4.2 AutoChooser Class Reference

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

Inheritance diagram for AutoChooser:



Classes

- struct entry_t

Public Member Functions

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

Protected Attributes

- size_t choice
- std::vector< entry_t > list

4.2.1 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.

4.2.2 Constructor & Destructor Documentation

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

4.2.3 Member Function Documentation

get_choice()

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

Get the currently selected auto choice

Returns

the identifier to the auto path

Return the selected autonomous

4.2.4 Member Data Documentation

choice

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

the current choice of auto

list

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

< a list of all possible auto choices

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

- auto_chooser.h
- auto_chooser.cpp

4.3 BasicSolenoidSet Class Reference

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

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.

4.3.1 Detailed Description

AutoCommand wrapper class for BasicSolenoidSet Using the Vex hardware functions

4.3.2 Constructor & Destructor Documentation

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)

4.3.3 Member Function Documentation

run()

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

Runs the BasicSolenoidSet Overrides run command from AutoCommand.

Returns

True Command runs once

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

- basic_command.h
- basic_command.cpp

4.4 BasicSpinCommand Class Reference

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

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.

4.4.1 Detailed Description

AutoCommand wrapper class for BasicSpinCommand using the vex hardware functions

4.4.2 Constructor & Destructor Documentation

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

4.4.3 Member Function Documentation

run()

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

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

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

- basic_command.h
- basic_command.cpp

4.5 BasicStopCommand Class Reference

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

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.

4.5.1 Detailed Description

AutoCommand wrapper class for BasicStopCommand Using the Vex hardware functions

4.5.2 Constructor & Destructor Documentation

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

4.5.3 Member Function Documentation

run()

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

Runs the BasicMotorStop Command Overrides run command from AutoCommand.

Runs the BasicMotorStop command Ovverides run command from AutoCommand.

Returns

True Command runs once

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

- basic_command.h
- basic_command.cpp

4.6 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>`

4.6.1 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.

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

- auto_command.h
- auto_command.cpp

4.7 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

4.7.1 Detailed Description

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

4.7.2 Constructor & Destructor Documentation

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

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

4.7.3 Member Function Documentation

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:

- screen.h
- screen.cpp

4.8 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 ()

4.8.1 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.

4.8.2 Constructor & Destructor Documentation

CommandController()

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

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

Parameters

cmds	

4.8.3 Member Function Documentation

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

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

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

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

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

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

run()

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

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

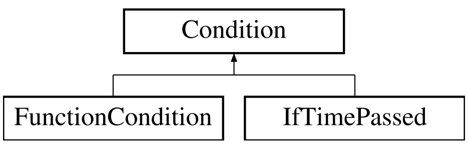
The documentation for this class was generated from the following files:

- command_controller.h
- command_controller.cpp

4.9 Condition Class Reference

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

Inheritance diagram for Condition:



4.9.1 Detailed Description

File: auto_command.h Desc: Interface for module-specifc commands A Condition is a function that returns true or false is_even is a predicate that would return true if a number is even For our purposes, a Condition is a choice to be made at runtime drive_sys.reached_point(10, 30) is a predicate time.has_elapsed(10, vex::seconds) is a predicate extend this class for different choices you wish to make

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

- auto_command.h
- auto_command.cpp

4.10 CustomEncoder Class Reference

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

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)

4.10.1 Detailed Description

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

4.10.2 Constructor & Destructor Documentation

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

4.10.3 Member Function Documentation

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

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

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

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

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:

- custom_encoder.h
- custom_encoder.cpp

4.11 DelayCommand Class Reference

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

Public Member Functions

- DelayCommand (int ms)
- bool run () override

4.11.1 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

4.11.2 Constructor & Destructor Documentation

DelayCommand()

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

Construct a delay command

Parameters

ms	the number of milliseconds to delay for

4.11.3 Member Function Documentation

run()

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

Delays for the amount of milliseconds stored in the command Overrides run from AutoCommand

Returns

true when complete

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

- delay_command.h

4.12 DriveForwardCommand Class Reference

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

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

4.12.1 Detailed Description

AutoCommand wrapper class for the drive_forward function in the TankDrive class

4.12.2 Constructor & Destructor Documentation

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

4.12.3 Member Function Documentation

on_timeout()

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

Cleans up drive system if we time out before finishing

reset the drive system if we timeout

run()

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

Run drive_forward Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

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

- drive_commands.h
- drive_commands.cpp

4.13 DriveStopCommand Class Reference

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

Public Member Functions

- DriveStopCommand (TankDrive &drive_sys)
- bool run () override

4.13.1 Detailed Description

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

4.13.2 Constructor & Destructor Documentation

DriveStopCommand()

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

Construct a DriveStop Command

Parameters

drive_sys	the drive system we are commanding

4.13.3 Member Function Documentation

run()

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

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

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

- drive_commands.h
- drive_commands.cpp

4.14 DriveToPointCommand Class Reference

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

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, Translation2d translation, directionType dir, double max_speed=1, double end_speed=0)
- bool run () override

4.14.1 Detailed Description

AutoCommand wrapper class for the drive_to_point function in the TankDrive class

4.14.2 Constructor & Destructor Documentation

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

DriveToPointCommand() [2/2]

```
DriveToPointCommand::DriveToPointCommand (
            TankDrive & drive_sys,
            Feedback & feedback,
            Translation2d translation,
            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
translation	the point to drive to
dir	the direction to drive
max_speed	0 -> 1 percentage of the drive systems speed to drive at

4.14.3 Member Function Documentation

run()

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

Run drive_to_point Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

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

- drive_commands.h
- drive_commands.cpp

4.15 AutoChooser::entry_t Struct Reference

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

Public Attributes

- std::string name

4.15.1 Detailed Description

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

4.15.2 Member Data Documentation

name

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

name of the auto repretsented by the block

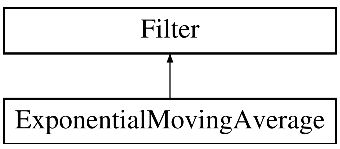
The documentation for this struct was generated from the following file:

- auto_chooser.h

4.16 ExponentialMovingAverage Class Reference

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

Inheritance diagram for ExponentialMovingAverage:



Public Member Functions

- ExponentialMovingAverage (int buffer_size)
- ExponentialMovingAverage (int buffer_size, double starting_value)
- void add_entry (double n) override
- double get_value () const override
- int get_size ()

4.16.1 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.

4.16.2 Constructor & Destructor Documentation

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

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

4.16.3 Member Function Documentation

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.

get_size()

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

How many samples the average is made from

Returns

the number of samples used to calculate this average

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.

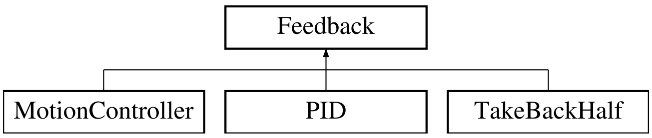
The documentation for this class was generated from the following files:

- moving_average.h
- moving_average.cpp

4.17 Feedback Class Reference

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

Inheritance diagram for Feedback:



Public Member Functions

- virtual void init (double start_pt, double set_pt)=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

4.17.1 Detailed Description

Interface so that subsystems can easily switch between feedback loops

Author

Ryan McGee

Date

9/25/2022

4.17.2 Member Function Documentation

get()

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

Returns

the last saved result from the feedback controller

Implemented in MotionController, PID, and TakeBackHalf.

init()

```
virtual void Feedback::init (
            double start_pt,
            double set_pt)  [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, PID, and TakeBackHalf.

is_on_target()

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

Returns

true if the feedback controller has reached it's setpoint

Implemented in MotionController, PID, and TakeBackHalf.

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 MotionController, PID, and TakeBackHalf.

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, and TakeBackHalf.

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

- feedback_base.h

4.18 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.

4.18.1 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

4.18.2 Constructor & Destructor Documentation

FeedForward()

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

Creates a FeedForward object.

Parameters

cfg	Configuration Struct for tuning

4.18.3 Member Function Documentation

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:

- feedforward.h

4.19 FeedForward::ff_config_t Struct Reference

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

Public Attributes

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

4.19.1 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

4.19.2 Member Data Documentation

kA

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

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

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.

kS

`double FeedForward::ff_config_t::kS`

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

kV

`double FeedForward::ff_config_t::kV`

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

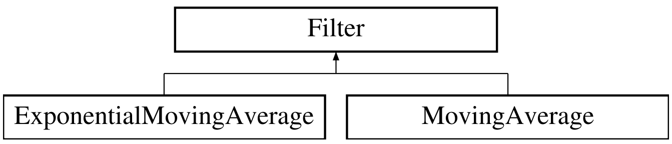
The documentation for this struct was generated from the following file:

- feedforward.h

4.20 Filter Class Reference

`#include <moving_average.h>`

Inheritance diagram for Filter:



4.20.1 Detailed Description

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

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

- moving_average.h

4.21 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

- int spinRPMTask (void ∗wheelPointer)

4.21.1 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

4.21.2 Constructor & Destructor Documentation

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

4.21.3 Member Function Documentation

get_motors()

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

Returns the motors

Returns

the motors used to run the flywheel

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

getRPM()

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

return the velocity of the flywheel

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

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()});

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

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

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

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

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

4.21.4 Friends And Related Symbol Documentation

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:

- flywheel.h
- flywheel.cpp

4.22 FlywheelStopCommand Class Reference

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

Public Member Functions

- FlywheelStopCommand (Flywheel &flywheel)
- bool run () override

4.22.1 Detailed Description

AutoCommand wrapper class for the stop function in the Flywheel class

4.22.2 Constructor & Destructor Documentation

FlywheelStopCommand()

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

Construct a FlywheelStopCommand

Parameters

flywheel	the flywheel system we are commanding

4.22.3 Member Function Documentation

run()

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

Run stop Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

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

- flywheel_commands.h
- flywheel_commands.cpp

4.23 FlywheelStopMotorsCommand Class Reference

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

Public Member Functions

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

4.23.1 Detailed Description

AutoCommand wrapper class for the stopMotors function in the Flywheel class

4.23.2 Constructor & Destructor Documentation

FlywheelStopMotorsCommand()

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

Construct a FlywheeStopMotors Command

Parameters

flywheel	the flywheel system we are commanding

4.23.3 Member Function Documentation

run()

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

Run stop Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

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

- flywheel_commands.h
- flywheel_commands.cpp

4.24 FlywheelStopNonTasksCommand Class Reference

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

4.24.1 Detailed Description

AutoCommand wrapper class for the stopNonTasks function in the Flywheel class

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

- flywheel_commands.h
- flywheel_commands.cpp

4.25 FunctionCommand Class Reference

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

4.25.1 Detailed Description

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

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

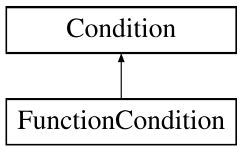
- auto_command.h

4.26 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:



4.26.1 Detailed Description

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

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

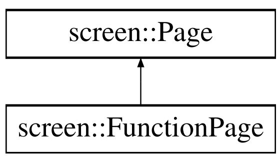
- auto_command.h
- auto_command.cpp

4.27 screen::FunctionPage Class Reference

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

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

Inheritance diagram for screen::FunctionPage:



Public Member Functions

- FunctionPage (update_func_t update_f, draw_func_t draw_t)

  Creates a function page.
- void update (bool was_pressed, int x, int y) override

  update uses the supplied update function to update this page
- void draw (vex::brain::lcd &, bool first_draw, unsigned int frame_number) override

  draw uses the supplied draw function to draw to the screen

4.27.1 Detailed Description

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

4.27.2 Constructor & Destructor Documentation

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

4.27.3 Member Function Documentation

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.

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:

- screen.h
- screen.cpp

4.28 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)

4.28.1 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

4.28.2 Member Function Documentation

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

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

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

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:

- generic_auto.h
- generic_auto.cpp

4.29 PurePursuit::hermite_point Struct Reference

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

4.29.1 Detailed Description

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

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

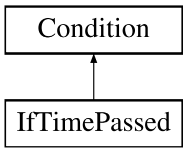
- pure_pursuit.h

4.30 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:



4.30.1 Detailed Description

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

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

- auto_command.h
- auto_command.cpp

4.31 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>`

4.31.1 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?

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

- auto_command.h
- auto_command.cpp

4.32 InterpolatingMap< KEY, VALUE > Class Template Reference

```
#include <interpolating_map.h>
```

Public Member Functions

- void insert (const KEY &key, const VALUE &value)
- VALUE operator[ ] (const KEY &key)
- void clear ()

4.32.1 Detailed Description

template<typename KEY, typename VALUE>
class InterpolatingMap< KEY, VALUE >

This class implements a map of key-value pairs.

If there is not a pair with the given key in the map, the value will be a linear interpolation of the preceding and following values.

Template Parameters

KEY	The type of the key.
VALUE	The type of the value.

4.32.2 Member Function Documentation

clear()

```
template<typename KEY, typename VALUE>
void InterpolatingMap< KEY, VALUE >::clear ( )  [inline]
```

Clears the contents of the map.

insert()

```
template<typename KEY, typename VALUE>
void InterpolatingMap< KEY, VALUE >::insert (
            const KEY & key,
            const VALUE & value)  [inline]
```

Inserts a key value pair.

Parameters

key	The key.
vlue	The value.

operator[]()

```
template<typename KEY, typename VALUE>
VALUE InterpolatingMap< KEY, VALUE >::operator[ ] (
            const KEY & key)  [inline]
```

Obtains the value at the given key.

If the key does not exactly match a pair in the map, it will interpolate between the preceding and following pairs.

Parameters

key	The key.

Returns

The value.

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

- interpolating_map.h

4.33 KalmanFilter< STATES, INPUTS, OUTPUTS > Class Template Reference

```
#include <kalman_filter.h>
```

Public Member Functions

- KalmanFilter (LinearSystem< STATES, INPUTS, OUTPUTS > &plant, const StateVector &state_stddevs, const OutputVector &measurement_stddevs)
- KalmanFilter (const StateMatrix &A, const InputMatrix &B, const EMat< OUTPUTS, STATES > &C, const EMat< OUTPUTS, INPUTS > &D, const StateVector &state_stddevs, const OutputVector &measurement_stddevs)
- StateMatrix P () const
- void set_P (const StateMatrix &P)
- const StateVector & xhat () const
- double xhat (int i) const
- void set_xhat (const StateVector &xhat)
- void set_xhat (int i, double value)
- void reset ()
- void predict (const InputVector &u, const double &dt)
- void correct (const OutputVector &y, const InputVector &u)
- void correct (const OutputVector &y, const InputVector &u, const EMat< OUTPUTS, OUTPUTS > &R)
- template<int ROWS>
  void correct (const EVec< ROWS > &y, const InputVector &u, const EMat< ROWS, STATES > &C, const EMat< ROWS, INPUTS > &D, const EMat< ROWS, ROWS > &R)

4.33.1 Detailed Description

template<int STATES, int INPUTS, int OUTPUTS>
class KalmanFilter< STATES, INPUTS, OUTPUTS >

Kalman filters combine predictions from a model and measurements to estimate a system's true state.

Each call of predict moves the state forward in time according to the matrix A, and the covariance has white noise Q added.

Each call of correct applies a measurement which moves the state more toward the true state, and it reduces the state covariance.

To read more about Kalman filters read: https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python

Template Parameters

STATES	Dimension of the state vector.
INPUTS	Dimension of the control input vector.
OUTPUTS	Dimension of the measurement vector.

4.33.2 Constructor & Destructor Documentation

KalmanFilter() [1/2]

```
template<int STATES, int INPUTS, int OUTPUTS>
KalmanFilter< STATES, INPUTS, OUTPUTS >::KalmanFilter (
            LinearSystem< STATES, INPUTS, OUTPUTS > & plant,
            const StateVector & state_stddevs,
            const OutputVector & measurement_stddevs)  [inline]
```

Constructs a Kalman filter.

Parameters

plant	The linear system the filter tracks.
state_stddevs	The standard deviations of the states.
measurement_stddevs	The standard deviations of the measurements.

KalmanFilter() [2/2]

```
template<int STATES, int INPUTS, int OUTPUTS>
KalmanFilter< STATES, INPUTS, OUTPUTS >::KalmanFilter (
            const StateMatrix & A,
            const InputMatrix & B,
            const EMat< OUTPUTS, STATES > & C,
            const EMat< OUTPUTS, INPUTS > & D,
            const StateVector & state_stddevs,
            const OutputVector & measurement_stddevs)  [inline]
```

Constructs a Kalman filter.

Parameters

A	The state matrix.
B	The input matrix.
C	The measurement matrix.
D	The feedthrough matrix.
state_stddevs	The standard deviations of the states.
measurement_stddevs	The standard deviations of the measurements.

4.33.3 Member Function Documentation

correct() [1/3]

```
template<int STATES, int INPUTS, int OUTPUTS>
template<int ROWS>
void KalmanFilter< STATES, INPUTS, OUTPUTS >::correct (
            const EVec< ROWS > & y,
            const InputVector & u,
            const EMat< ROWS, STATES > & C,
            const EMat< ROWS, INPUTS > & D,
            const EMat< ROWS, ROWS > & R)  [inline]
```

Correct the state estimate using the measurements in y, custom measurement and feedthrough matrices, and custom measurement measurement noise. This is useful for when a different set of measurements are being applied than what the plant defines.

Parameters

y	The vector of measurements.
u	The control input used in the last predict step.
C	The measurement matrix to use for this step.
D	The feedthrough matrix to use for this step.
R	The measurement noise matrix to use for this step.

correct() [2/3]

```
template<int STATES, int INPUTS, int OUTPUTS>
void KalmanFilter< STATES, INPUTS, OUTPUTS >::correct (
            const OutputVector & y,
            const InputVector & u)  [inline]
```

Correct the state estimate using the measurements in y.

Parameters

y	The vector of measurements.
u	The control input used in the last predict step.

correct() [3/3]

```
template<int STATES, int INPUTS, int OUTPUTS>
void KalmanFilter< STATES, INPUTS, OUTPUTS >::correct (
            const OutputVector & y,
            const InputVector & u,
            const EMat< OUTPUTS, OUTPUTS > & R)  [inline]
```

Correct the state estimate using the measurements in y, and custom measurement noise matrix. This is useful for when the noise in the measurements vary.

Parameters

y	The vector of measurements.
u	The control input used in the last predict step.
R	The measurement noise matrix to use for this step.

P()

```
template<int STATES, int INPUTS, int OUTPUTS>
StateMatrix KalmanFilter< STATES, INPUTS, OUTPUTS >::P () const  [inline]
```

Returns the covariance matrix P.

predict()

```
template<int STATES, int INPUTS, int OUTPUTS>
void KalmanFilter< STATES, INPUTS, OUTPUTS >::predict (
            const InputVector & u,
            const double & dt)  [inline]
```

Projects the state into the future by dt seconds with control input u.

Parameters

u	The control input.
dt	The timestep in seconds.

reset()

```
template<int STATES, int INPUTS, int OUTPUTS>
void KalmanFilter< STATES, INPUTS, OUTPUTS >::reset ()  [inline]
```

Resets the filter.

set_P()

```
template<int STATES, int INPUTS, int OUTPUTS>
void KalmanFilter< STATES, INPUTS, OUTPUTS >::set_P (
            const StateMatrix & P)  [inline]
```

Set the current covariance matrix P.

Parameters

P	The covariance matrix P.

set_xhat() [1/2]

```
template<int STATES, int INPUTS, int OUTPUTS>
void KalmanFilter< STATES, INPUTS, OUTPUTS >::set_xhat (
            const StateVector & xhat)  [inline]
```

Set the current state estimate x-hat.

set_xhat() [2/2]

```
template<int STATES, int INPUTS, int OUTPUTS>
void KalmanFilter< STATES, INPUTS, OUTPUTS >::set_xhat (
            int i,
            double value)  [inline]
```

Set one element of the current state estimate x-hat.

Parameters

i	Row of x-hat.

xhat() [1/2]

```
template<int STATES, int INPUTS, int OUTPUTS>
const StateVector & KalmanFilter< STATES, INPUTS, OUTPUTS >::xhat () const  [inline]
```

Returns the current state estimate x-hat.

xhat() [2/2]

```
template<int STATES, int INPUTS, int OUTPUTS>
double KalmanFilter< STATES, INPUTS, OUTPUTS >::xhat (
            int i) const  [inline]
```

Returns one element of the current state estimate x-hat.

Parameters

i	Row of x-hat.

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

- kalman_filter.h

4.34 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))

4.34.1 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

4.34.2 Constructor & Destructor Documentation

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

4.34.3 Member Function Documentation

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

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

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).

get_async()

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

Returns

whether or not the background thread is running the lift

get_setpoint()

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

Returns

The current setpoint for the lift

hold()

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

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

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.

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

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

set_sensor_function()

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

set_sensor_reset()

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

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:

- lift.h

4.35 Lift< T >::lift_cfg_t Struct Reference

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

4.35.1 Detailed Description

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

lift_cfg_t holds the physical parameter specifications of a lify system. includes:

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

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

- lift.h

4.36 LinearPlantInversionFeedforward< STATES, INPUTS > Class Template Reference

```
#include <linear_plant_inversion_feedforward.h>
```

Public Member Functions

- template<int OUTPUTS>
  LinearPlantInversionFeedforward (LinearSystem< STATES, INPUTS, OUTPUTS > &plant, const double &dt)
- LinearPlantInversionFeedforward (const EMat< STATES, STATES > &A, const EMat< STATES, INPUTS > &B, const double &dt)
- EVec< INPUTS > calculate (const EVec< STATES > &r, const EVec< STATES > &next_r)
- EVec< INPUTS > calculate (const EVec< STATES > &next_r)
- EVec< INPUTS > calculate (const EVec< STATES > &r, const EVec< STATES > &next_r, const double &dt)
- EVec< INPUTS > calculate (const EVec< STATES > &next_r, const double &dt)
- void reset (const EVec< STATES > &initial_state)
- void reset ()
- void set_r (const EVec< STATES > &r)

4.36.1 Detailed Description

template<int STATES, int INPUTS>
class LinearPlantInversionFeedforward< STATES, INPUTS >

This class computes a feedforward control input by inverting the discrete plant dynamics. A continuous linear system is provided, it is then discretized on some timestep, then the feedforward control input is computed to satisfy:

B_d * u_ff = next_state - A_d * current_state

4.36.2 Constructor & Destructor Documentation

LinearPlantInversionFeedforward() [1/2]

```
template<int STATES, int INPUTS>
template<int OUTPUTS>
LinearPlantInversionFeedforward< STATES, INPUTS >::LinearPlantInversionFeedforward (
            LinearSystem< STATES, INPUTS, OUTPUTS > & plant,
            const double & dt)  [inline]
```

Constructs a feedforward given a plant and the nominal timestep.

Template Parameters

OUTPUTS	The number of outputs of the plant.

Parameters

plant	The linear system.
dt	The nominal timestep in seconds.

LinearPlantInversionFeedforward() [2/2]

```
template<int STATES, int INPUTS>
LinearPlantInversionFeedforward< STATES, INPUTS >::LinearPlantInversionFeedforward (
            const EMat< STATES, STATES > & A,
            const EMat< STATES, INPUTS > & B,
            const double & dt)  [inline]
```

Construts a feedforward given the state and input matrices from a plant.

Parameters

A	The state matrix of the linear system.
B	The input matrix of the linear system.
dt	The nominal timestep in seconds.

4.36.3 Member Function Documentation

calculate() [1/4]

```
template<int STATES, int INPUTS>
EVec< INPUTS > LinearPlantInversionFeedforward< STATES, INPUTS >::calculate (
            const EVec< STATES > & next_r)  [inline]
```

Computes the feedforward control input given only the next reference state. This assumes that the previous reference is already set.

Parameters

next_r	The next reference state.

calculate() [2/4]

```
template<int STATES, int INPUTS>
EVec< INPUTS > LinearPlantInversionFeedforward< STATES, INPUTS >::calculate (
            const EVec< STATES > & next_r,
            const double & dt)  [inline]
```

Computes the feedforward control input given only the next reference state. This assumes that the previous reference is already set.

This is slower because it discretizes A and B on each run, requiring computing a matrix exponential. Don't use this unless you have to.

Parameters

next_r	The next reference state.
dt	The timestep for this run.

calculate() [3/4]

```
template<int STATES, int INPUTS>
EVec< INPUTS > LinearPlantInversionFeedforward< STATES, INPUTS >::calculate (
            const EVec< STATES > & r,
            const EVec< STATES > & next_r)  [inline]
```

Computes the feedforward control input given the current reference state and the next reference state. This also sets the current reference state to the next reference state for you.

Parameters

r	The current reference state.
next_r	The next reference state.

calculate() [4/4]

```
template<int STATES, int INPUTS>
EVec< INPUTS > LinearPlantInversionFeedforward< STATES, INPUTS >::calculate (
            const EVec< STATES > & r,
            const EVec< STATES > & next_r,
            const double & dt)  [inline]
```

Computes the feedforward control input given the current reference state and the next reference state. This also sets the current reference state to the next reference state for you. Use this function if your timestep is not the same between each run.

This is slower because it discretizes A and B on each run, requiring computing a matrix exponential. Don't use this unless you have to.

Parameters

r	The current reference state.
next←_r	The next reference state.
dt	The timestep for this run.

reset() [1/2]

```
template<int STATES, int INPUTS>
void LinearPlantInversionFeedforward< STATES, INPUTS >::reset ()  [inline]
```

Resets the reference to all zeros, and the feedforward to zero.

reset() [2/2]

```
template<int STATES, int INPUTS>
void LinearPlantInversionFeedforward< STATES, INPUTS >::reset (
            const EVec< STATES > & initial_state)  [inline]
```

Resets the reference to the given state, and the feedforward to zero.

Parameters

initial_state	The state to set the current reference to.

set_r()

```
template<int STATES, int INPUTS>
void LinearPlantInversionFeedforward< STATES, INPUTS >::set_r (
            const EVec< STATES > & r)  [inline]
```

Sets the current reference to a given state.

Parameters

r	The state to set the current reference to.

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

- linear_plant_inversion_feedforward.h

4.37 LinearQuadraticRegulator< STATES, INPUTS > Class Template Reference

```
#include <linear_quadratic_regulator.h>
```

Public Member Functions

- template<int OUTPUTS>
  LinearQuadraticRegulator (LinearSystem< STATES, INPUTS, OUTPUTS > &plant, const VectorX &Qtolerances, const VectorU &Rtolerances, const double &dt)
- LinearQuadraticRegulator (const MatrixA &A, const MatrixB &B, const VectorX &Qtolerances, const VectorU &Rtolerances, const double &dt)
- LinearQuadraticRegulator (const MatrixA &A, const MatrixB &B, const EMat< STATES, STATES > &Q, const EMat< INPUTS, INPUTS > &R, const double &dt)
- VectorU calculate (const VectorX &x, const VectorX &r)
- template<int OUTPUTS>
  void latency_compensate (LinearSystem< STATES, INPUTS, OUTPUTS > &plant, const double &dt, const double &input_delay)

4.37.1 Detailed Description

template<int STATES, int INPUTS>
class LinearQuadraticRegulator< STATES, INPUTS >

Class implements an LQR controller. This finds the optimal gain matrix K where:

u = K(r - x)

K is optimized to minimize a cost function:

∞

J = Σ $x_k{}^T Q x_k$ + $u_k{}^T R u_k$ k=0

Where Q and R are the state and control cost matrices.

Template Parameters

STATES	The number of states in the system.
INPUTS	The number of inputs to the system.

4.37.2 Constructor & Destructor Documentation

LinearQuadraticRegulator() [1/3]

```
template<int STATES, int INPUTS>
template<int OUTPUTS>
LinearQuadraticRegulator< STATES, INPUTS >::LinearQuadraticRegulator (
            LinearSystem< STATES, INPUTS, OUTPUTS > & plant,
            const VectorX & Qtolerances,
            const VectorU & Rtolerances,
            const double & dt)  [inline]
```

Constructs an LQR given a plant, a vector of tolerances for the states and inputs, and the timestep in seconds.

Template Parameters

OUTPUTS	The number of outputs of the plant.

Parameters

plant	The linear system to control.
Qtolerances	A vector of tolerances for each state.
Rtolerances	A vector of tolerances for each input.

LinearQuadraticRegulator() [2/3]

```
template<int STATES, int INPUTS>
LinearQuadraticRegulator< STATES, INPUTS >::LinearQuadraticRegulator (
            const MatrixA & A,
            const MatrixB & B,
            const VectorX & Qtolerances,
            const VectorU & Rtolerances,
            const double & dt) [inline]
```

Constructs an LQR given state and input matrices, a vector of tolerances for the states and inputs, and the timestep in seconds.

Parameters

A	The state matrix of the linear system.
B	The input matrix of the linear system.
Qtolerances	A vector of tolerances for each state.
Rtolerances	A vector of tolerances for each input.

LinearQuadraticRegulator() [3/3]

```
template<int STATES, int INPUTS>
LinearQuadraticRegulator< STATES, INPUTS >::LinearQuadraticRegulator (
            const MatrixA & A,
            const MatrixB & B,
            const EMat< STATES, STATES > & Q,
            const EMat< INPUTS, INPUTS > & R,
            const double & dt) [inline]
```

Constructs an LQR given state and input matrices, the cost matrices of states and inputs, and the timestep in seconds.

Parameters

A	The state matrix of the linear system.
B	The input matrix of the linear system.
Q	The cost matrix of the states.
R	The cost matrix of the inputs.

4.37.3 Member Function Documentation

calculate()

```
template<int STATES, int INPUTS>
VectorU LinearQuadraticRegulator< STATES, INPUTS >::calculate (
            const VectorX & x,
            const VectorX & r)  [inline]
```

Computes the control input u as:

u = K(r - x)

Parameters

x	The current state.
r	The reference state.

latency_compensate()

```
template<int STATES, int INPUTS>
template<int OUTPUTS>
void LinearQuadraticRegulator< STATES, INPUTS >::latency_compensate (
            LinearSystem< STATES, INPUTS, OUTPUTS > & plant,
            const double & dt,
            const double & input_delay)  [inline]
```

Recomputes K to work for a time delayed state.

Kdelay = K(A - BK)^(delay / dt)

Template Parameters

OUTPUTS	The number of outputs of the plant

Parameters

plant	The linear system.
dt	The timestep in seconds.
input_delay	The time delay of the system.

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

- linear_quadratic_regulator.h

4.38 LinearSystem< STATES, INPUTS, OUTPUTS > Class Template Reference

```
#include <linear_system.h>
```

Public Member Functions

- LinearSystem (const MatrixA &A, const MatrixB &B, const MatrixC &C, const MatrixD &D)
- MatrixA A ()
- MatrixB B ()
- const std::tuple< std::tuple< MatrixA, MatrixB > > & discAB (const double &dt)
- MatrixC C ()
- MatrixD D ()
- VectorX compute_X (const VectorX &x, const VectorU &u, double dt)
- VectorY compute_Y (const VectorX &x, const VectorU &u)

4.38.1 Detailed Description

template<int STATES, int INPUTS, int OUTPUTS>
class LinearSystem< STATES, INPUTS, OUTPUTS >

This class represents a state-space model of a linear system.

It contains the following continuous matrices: A, System matrix B, Input matrix C, Output matrix D, Feedthrough matrix

4.38.2 Constructor & Destructor Documentation

LinearSystem()

```
template<int STATES, int INPUTS, int OUTPUTS>
LinearSystem< STATES, INPUTS, OUTPUTS >::LinearSystem (
            const MatrixA & A,
            const MatrixB & B,
            const MatrixC & C,
            const MatrixD & D)  [inline]
```

Constructs a discrete linear system with the given continuous matrices.

Parameters

A	The continuous system matrix
B	The continuous input matrix
C	The output matrix
D	The feedthrough matrix

4.38.3 Member Function Documentation

A()

```
template<int STATES, int INPUTS, int OUTPUTS>
MatrixA LinearSystem< STATES, INPUTS, OUTPUTS >::A ()  [inline]
```

Returns the continuous system matrix A.

B()

```
template<int STATES, int INPUTS, int OUTPUTS>
MatrixB LinearSystem< STATES, INPUTS, OUTPUTS >::B ()  [inline]
```

Returns the continuous input matrix B.

C()

```
template<int STATES, int INPUTS, int OUTPUTS>
MatrixC LinearSystem< STATES, INPUTS, OUTPUTS >::C ()  [inline]
```

Returns the output matrix C.

compute_X()

```
template<int STATES, int INPUTS, int OUTPUTS>
VectorX LinearSystem< STATES, INPUTS, OUTPUTS >::compute_X (
            const VectorX & x,
            const VectorU & u,
            double dt)  [inline]
```

Computes the new state vector given the previous state vector, an input vector, and the timestep in seconds.

Parameters

x	The current state vector.
u	The input vector.
dt	The timestep in seconds.

Returns

The new state vector.

compute_Y()

```
template<int STATES, int INPUTS, int OUTPUTS>
VectorY LinearSystem< STATES, INPUTS, OUTPUTS >::compute_Y (
            const VectorX & x,
            const VectorU & u)  [inline]
```

Computes the output vector given a state and an input.

Parameters

x	The state vector.
u	The input vector.

Returns

The output vector.

D()

```
template<int STATES, int INPUTS, int OUTPUTS>
MatrixD LinearSystem< STATES, INPUTS, OUTPUTS >::D ()  [inline]
```

Returns the feedthrough matrix D.

discAB()

```
template<int STATES, int INPUTS, int OUTPUTS>
const std::tuple< std::tuple< MatrixA, MatrixB > > & LinearSystem< STATES, INPUTS, OUTPUTS
>::discAB (
            const double & dt)  [inline]
```

Returns a tuple of A and B after being discretized.

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

- linear_system.h

4.39 Logger Class Reference

Class to simplify writing to files.

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

Public Member Functions

- Logger (const std::string &filename)

    Create a logger that will save to a file.
- **Logger** (const Logger &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

4.39.1 Detailed Description

Class to simplify writing to files.

4.39.2 Constructor & Destructor Documentation

Logger()

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

Create a logger that will save to a file.

Parameters

filename	the file to save to

4.39.3 Member Function Documentation

Log() [1/2]

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

Write a string to the log.

Parameters

s	the string to write

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

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

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

Logln() [1/2]

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

Write a string and newline to the log.

Parameters

s	the string to write

Logln() [2/2]

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

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

Parameters

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

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

- logger.h
- logger.cpp

4.40 MotionController::m_profile_cfg_t Struct Reference

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

Public Attributes

- double **max_v**

  the maximum velocity the robot can drive
- double **accel**

  the most acceleration the robot can do
- PID::pid_config_t **pid_cfg**

  configuration parameters for the internal PID controller
- FeedForward::ff_config_t **ff_cfg**

  configuration parameters for the internal

4.40.1 Detailed Description

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

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

- motion_controller.h

4.41 StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage Class Reference

MaybeMessage a message of Message type or nothing MaybeMessage m = {}; // empty MaybeMessage m = Message::EnumField1.

```
#include <state_machine.h>
```

Public Member Functions

- **MaybeMessage** ()

  Empty message - when theres no message.
- MaybeMessage (Message msg)

  Create a maybemessage with a message.
- bool has_message ()

  check if the message is here
- Message message ()

  Get the message stored. The return value is invalid unless has_message returned true.

4.41.1 Detailed Description

template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log = false> class StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage

MaybeMessage a message of Message type or nothing MaybeMessage m = {}; // empty MaybeMessage m = Message::EnumField1.

4.41.2 Constructor & Destructor Documentation

MaybeMessage()

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log =
false>
StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage::MaybeMessage (
            Message msg )  [inline]
```

Create a maybemessage with a message.

Parameters

msg	the message to hold on to

4.41.3 Member Function Documentation

has_message()

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log =
false>
bool StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage::has_message ()
[inline]
```

check if the message is here

Returns

true if there is a message

message()

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log =
false>
Message StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage::message ()
[inline]
```

Get the message stored. The return value is invalid unless has_message returned true.

Returns

The message if it exists. Undefined otherwise

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

- state_machine.h

4.42 MecanumDrive Class Reference

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

Classes

- struct mecanumdrive_config_t

Public Member Functions

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

4.42.1 Detailed Description

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

4.42.2 Constructor & Destructor Documentation

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

4.42.3 Member Function Documentation

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.

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

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)

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:

- mecanum_drive.h
- mecanum_drive.cpp

4.43 MecanumDrive::mecanumdrive_config_t Struct Reference

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

4.43.1 Detailed Description

Configure the Mecanum drive PID tunings and robot configurations

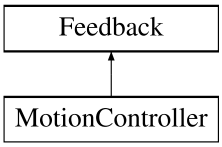
The documentation for this struct was generated from the following file:

- mecanum_drive.h

4.44 MotionController Class Reference

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

Inheritance diagram for MotionController:



Classes

- struct m_profile_cfg_t

Public Member Functions

- MotionController (m_profile_cfg_t &config)

    Construct a new Motion Controller object.
- void init (double start_pt, double end_pt) 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

Static Public Member Functions

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

4.44.1 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

4.44.2 Constructor & Destructor Documentation

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

4.44.3 Member Function Documentation

get()

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

Returns

the last saved result from the feedback controller

Implements Feedback.

get_motion()

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

Returns

The current postion, velocity and acceleration setpoints

init()

```
void MotionController::init (
            double start_pt,
            double end_pt) [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

Implements Feedback.

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.

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.

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

update()

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

Update the motion profile with a new sensor value.

Parameters

sensor_val	Value from the sensor

Returns

the motor input generated from the motion profile

Implements Feedback.

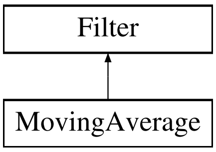
The documentation for this class was generated from the following files:

- motion_controller.h
- motion_controller.cpp

4.45 MovingAverage Class Reference

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

Inheritance diagram for MovingAverage:



Public Member Functions

- MovingAverage (int buffer_size)
- MovingAverage (int buffer_size, double starting_value)
- void add_entry (double n) override
- double get_value () const override
- int get_size () const

4.45.1 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).

4.45.2 Constructor & Destructor Documentation

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

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

4.45.3 Member Function Documentation

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.

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

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.

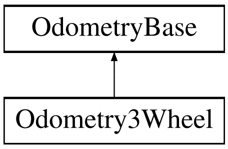
The documentation for this class was generated from the following files:

- moving_average.h
- moving_average.cpp

4.46 Odometry3Wheel Class Reference

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

Inheritance diagram for Odometry3Wheel:



Classes

- struct odometry3wheel_cfg_t

Public Member Functions

- Odometry3Wheel (CustomEncoder &lside_fwd, CustomEncoder &rside_fwd, CustomEncoder &off_axis, odometry3wheel_cfg_t &cfg, bool is_async=true)
- Pose2d update () override
- void tune (vex::controller &con, TankDrive &drive)

Public Member Functions inherited from OdometryBase

- OdometryBase (bool is_async)
- virtual Pose2d get_position (void)
- virtual void set_position (const Pose2d &newpos=zero_pos)
- void end_async ()
- virtual double get_speed ()
- virtual 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 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
- vex::task * handle
- vex::mutex mut
- Pose2d current_pos
- double speed
- double accel
- double ang_speed_deg
- double ang_accel_deg

4.46.1 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

4.46.2 Constructor & Destructor Documentation

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

4.46.3 Member Function Documentation

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

update()

```
Pose2d 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:

- odometry_3wheel.h
- odometry_3wheel.cpp

4.47 Odometry3Wheel::odometry3wheel_cfg_t Struct Reference

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

Public Attributes

- double wheelbase_dist
- double off_axis_center_dist
- double wheel_diam

4.47.1 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

4.47.2 Member Data Documentation

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

wheel_diam

`double Odometry3Wheel::odometry3wheel_cfg_t::wheel_diam`

the diameter of the tracking wheel

wheelbase_dist

`double Odometry3Wheel::odometry3wheel_cfg_t::wheelbase_dist`

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

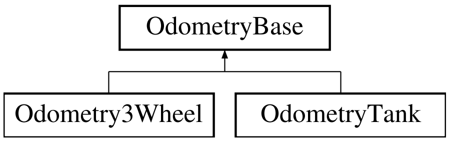
The documentation for this struct was generated from the following file:

- odometry_3wheel.h

4.48 OdometryBase Class Reference

`#include <odometry_base.h>`

Inheritance diagram for OdometryBase:



Public Member Functions

- OdometryBase (bool is_async)
- virtual Pose2d get_position (void)
- virtual void set_position (const Pose2d &newpos=zero_pos)
- virtual Pose2d update ()=0
- void end_async ()
- virtual double get_speed ()
- virtual 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 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
- vex::task ∗ handle
- vex::mutex mut
- Pose2d current_pos
- double speed
- double accel
- double ang_speed_deg
- double ang_accel_deg

4.48.1 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

4.48.2 Constructor & Destructor Documentation

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

4.48.3 Member Function Documentation

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.

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.

get_accel()

```
double OdometryBase::get_accel () [virtual]
```

Get the current acceleration

Returns

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

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)

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)

get_position()

```
Pose2d OdometryBase::get_position (
             void ) [virtual]
```

Gets the current position and rotation

Returns

the position that the odometry believes the robot is at

Gets the current position and rotation

get_speed()

```
double OdometryBase::get_speed () [virtual]
```

Get the current speed

Returns

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

set_position()

```
void OdometryBase::set_position (
             const Pose2d & 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.

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.

update()

```
virtual Pose2d 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.

4.48.4 Member Data Documentation

accel

```
double OdometryBase::accel
```

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

ang_accel_deg

```
double OdometryBase::ang_accel_deg
```

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

ang_speed_deg

```
double OdometryBase::ang_speed_deg
```

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

current_pos

```
Pose2d OdometryBase::current_pos
```

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

handle

```
vex::task* OdometryBase::handle
```

handle to the vex task that is running the odometry code

mut

```
vex::mutex OdometryBase::mut
```

Mutex to control multithreading

speed

```
double OdometryBase::speed
```

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

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

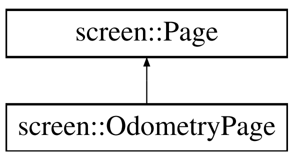
- odometry_base.h
- odometry_base.cpp

4.49 screen::OdometryPage Class Reference

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

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

Inheritance diagram for screen::OdometryPage:



Public Member Functions

- OdometryPage (OdometryBase &odom, double robot_width, double robot_height, bool do_trail)

    Create an odometry trail. Make sure odometry is initilized before now.
- void update (bool was_pressed, int x, int y) override
- void draw (vex::brain::lcd &, bool first_draw, unsigned int frame_number) override

4.49.1 Detailed Description

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

4.49.2 Constructor & Destructor Documentation

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

4.49.3 Member Function Documentation

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.

update()

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

See also

Page::update

Reimplemented from screen::Page.

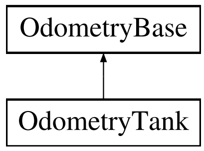
The documentation for this class was generated from the following files:

- screen.h
- screen.cpp

4.50 OdometryTank Class Reference

```
#include <odometry_tank.h>
```

Inheritance diagram for OdometryTank:



Public Member Functions

- OdometryTank (vex::motor_group &left_side, vex::motor_group &right_side, robot_specs_t &config, vex::inertial *imu=NULL, bool is_async=true)
- OdometryTank (CustomEncoder &left_custom_enc, CustomEncoder &right_custom_enc, robot_specs_t &config, vex::inertial *imu=NULL, bool is_async=true)
- OdometryTank (vex::encoder &left_vex_enc, vex::encoder &right_vex_enc, robot_specs_t &config, vex::inertial *imu=NULL, bool is_async=true)
- Pose2d update () override
- void set_position (const Pose2d &newpos=zero_pos) override

Public Member Functions inherited from OdometryBase

- OdometryBase (bool is_async)
- virtual Pose2d get_position (void)
- void end_async ()
- virtual double get_speed ()
- virtual 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 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
- vex::task * handle
- vex::mutex mut
- Pose2d current_pos
- double speed
- double accel
- double ang_speed_deg
- double ang_accel_deg

4.50.1 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.

4.50.2 Constructor & Destructor Documentation

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().

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().

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().

4.50.3 Member Function Documentation

set_position()

```
void OdometryTank::set_position (
            const Pose2d & 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.

update()

```
Pose2d 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:

- odometry_tank.h
- odometry_tank.cpp

4.51 OdomSetPosition Class Reference

`#include <drive_commands.h>`

Public Member Functions

- OdomSetPosition (OdometryBase &odom, const Pose2d &newpos=OdometryBase::zero_pos)
- bool run () override

4.51.1 Detailed Description

AutoCommand wrapper class for the set_position function in the Odometry class

4.51.2 Constructor & Destructor Documentation

OdomSetPosition()

```
OdomSetPosition::OdomSetPosition (
            OdometryBase & odom,
            const Pose2d & 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

4.51.3 Member Function Documentation

run()

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

Run set_position Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

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

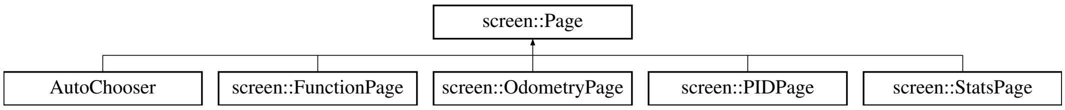
- drive_commands.h
- drive_commands.cpp

4.52 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)

4.52.1 Detailed Description

Page describes one part of the screen slideshow.

4.52.2 Member Function Documentation

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 screen::FunctionPage, screen::OdometryPage, screen::PIDPage, and screen::StatsPage.

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 screen::FunctionPage, screen::OdometryPage, screen::PIDPage, and screen::StatsPage.

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

- screen.h

4.53 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>
```

4.53.1 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.

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

- auto_command.h
- auto_command.cpp

4.54 PurePursuit::Path Class Reference

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

Public Member Functions

- Path (std::vector< Translation2d > points, double radius)
- const std::vector< Translation2d > get_points ()
- double get_radius ()
- bool is_valid ()

4.54.1 Detailed Description

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

4.54.2 Constructor & Destructor Documentation

Path()

```
PurePursuit::Path::Path (
        std::vector< Translation2d > points,
        double radius)
```

Create a Path

Parameters

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

4.54.3 Member Function Documentation

get_points()

```
const std::vector< Translation2d > PurePursuit::Path::get_points ()
```

Get the points associated with this Path

get_radius()

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

Get the radius associated with this Path

is_valid()

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

Get whether this path will behave as expected

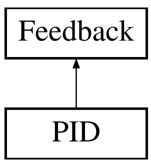
The documentation for this class was generated from the following files:

- pure_pursuit.h
- pure_pursuit.cpp

4.55 PID Class Reference

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

Inheritance diagram for PID:



Classes

- struct pid_config_t

Public Types

- enum ERROR_TYPE

Public Member Functions

- PID (pid_config_t &config)
- void init (double start_pt, double set_pt) override
- double update (double sensor_val) override
- double update (double sensor_val, double v_setpt)
- 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

4.55.1 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

4.55.2 Member Enumeration Documentation

ERROR_TYPE

```
enum PID::ERROR_TYPE
```

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

4.55.3 Constructor & Destructor Documentation

PID()

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

Create the PID object

Parameters

config	the configuration data for this controller

Create the PID object

4.55.4 Member Function Documentation

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.

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

get_sensor_val()

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

gets the sensor value that we were last updated with

Returns

sensor_val

get_target()

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

Get the PID's target

Returns

the target the PID controller is trying to achieve

init()

```
void PID::init (
            double start_pt,
            double set_pt)  [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	completely 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.

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.

reset()

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

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

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.

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

update() [1/2]

```
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.

update() [2/2]

```
double PID::update (
            double sensor_val,
            double v_setpt)
```

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
v_setpt	Expected velocity setpoint, to subtract from the D term (for velocity control)

Returns

the new output. What would be returned by PID::get()

4.55.5 Member Data Documentation

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:

- pid.h
- pid.cpp

4.56 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

4.56.1 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

4.56.2 Member Data Documentation

error_method

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

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

on_target_time

```
double PID::pid_config_t::on_target_time
```

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

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

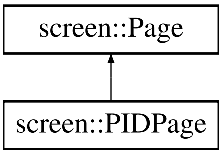
- pid.h

4.57 screen::PIDPage Class Reference

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

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

Inheritance diagram for screen::PIDPage:



Public Member Functions

- PIDPage (PID &pid, std::string name, std::function< void(void)> onchange=[ ]() {})
    Create a PIDPage.
- void update (bool was_pressed, int x, int y) override
- void draw (vex::brain::lcd &, bool first_draw, unsigned int frame_number) override

4.57.1 Detailed Description

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

4.57.2 Constructor & Destructor Documentation

PIDPage()

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

Create a PIDPage.

Parameters

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

4.57.3 Member Function Documentation

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.

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:

- screen.h
- screen.cpp

4.58 Pose2d Class Reference

```
#include <pose2d.h>
```

Public Member Functions

- constexpr Pose2d ()
- Pose2d (const Translation2d &translation, const Rotation2d &rotation)
- Pose2d (const double &x, const double &y, const Rotation2d &rotation)
- Pose2d (const double &x, const double &y, const double &radians)
- Pose2d (const Translation2d &translation, const double &radians)
- Pose2d (const Eigen::Vector3d &pose_vector)
- Translation2d translation () const
- double x () const
- double y () const
- Rotation2d rotation () const
- void setRotationRad (double rotRad)
- void setRotationDeg (double rotDeg)
- bool operator== (const Pose2d other) const
- Pose2d operator* (const double &scalar) const
- Pose2d operator/ (const double &scalar) const
- Pose2d operator+ (const Transform2d &transform) const
- Transform2d operator- (const Pose2d &other) const
- Pose2d relative_to (const Pose2d &other) const
- Pose2d transform_by (const Transform2d &transform) const
- Pose2d exp (const Twist2d &twist) const
- Twist2d log (const Pose2d &end_pose) const

Friends

- std::ostream & operator<< (std::ostream &os, const Pose2d &pose)

4.58.1 Detailed Description

Class representing a pose in 2d space with x, y, and rotational components

Assumes conventional cartesian coordinate system: Looking down at the coordinate plane, +X is right +Y is up +Theta is counterclockwise

4.58.2 Constructor & Destructor Documentation

Pose2d() [1/6]

```
Pose2d::Pose2d ()  [inline], [constexpr]
```

Default Constructor for Pose2d

Pose2d() [2/6]

```
Pose2d::Pose2d (
            const Translation2d & translation,
            const Rotation2d & rotation)
```

Constructs a pose with given translation and rotation components.

Parameters

translation	translational component.
rotation	rotational component.

Pose2d() [3/6]

```
Pose2d::Pose2d (
            const double & x,
            const double & y,
            const Rotation2d & rotation)
```

Constructs a pose with given translation and rotation components.

Parameters

x	x component.
y	y component.
rotation	rotational component.

Pose2d() [4/6]

```
Pose2d::Pose2d (
            const double & x,
            const double & y,
            const double & radians)
```

Constructs a pose with given translation and rotation components.

Parameters

x	x component.
y	y component.
radians	rotational component in radians.

Pose2d() [5/6]

```
Pose2d::Pose2d (
            const Translation2d & translation,
            const double & radians)
```

Constructs a pose with given translation and rotation components.

Parameters

translation	translational component.
radians	rotational component in radians.

Pose2d() [6/6]

```
Pose2d::Pose2d (
            const Eigen::Vector3d & pose_vector)
```

Constructs a pose with given translation and rotation components.

Parameters

pose_vector	vector of the form [x, y, theta].

4.58.3 Member Function Documentation

exp()

```
Pose2d Pose2d::exp (
            const Twist2d & twist) const
```

Applies a twist (pose delta) to a pose by including first order dynamics of heading.

When applying a twist, imagine a constant angular velocity, the translational components must be rotated into the global frame at every point along the twist, simply adding the deltas does not do this, and using euler integration results in some error. This is the analytic solution that that problem.

Can also be thought of more simply as applying a twist as following an arc rather than a straight line.

See this document for more information on the pose exponential and its derivation. https://file.tavsys.net/control/controls-engineering-in-frc.pdf#section.10.2

Parameters

old_pose	The pose to which the twist will be applied.
twist	The twist, represents a pose delta.

Returns

new pose that has been moved forward according to the twist.

log()

```
Twist2d Pose2d::log (
            const Pose2d & end_pose) const
```

The inverse of the pose exponential.

Determines the twist required to go from this pose to the given end pose. suppose you have Pose2d a, Twist2d twist if a.exp(twist) = b then a.log(b) = twist

Parameters

end_pose	the end pose to find the mapping to.

Returns

the twist required to go from this pose to the given end

operator*()

```
Pose2d Pose2d::operator* (
            const double & scalar) const
```

Multiplies this pose by a scalar. Simply multiplies each component.

Parameters

scalar	the scalar value to multiply by.

operator+()

```
Pose2d Pose2d::operator+ (
            const Transform2d & transform) const
```

Adds a transform to this pose. Transforms the pose in the pose's frame.

Parameters

transform	the change in pose.

operator-()

```
Transform2d Pose2d::operator- (
            const Pose2d & other) const
```

Subtracts one pose from another to find the transform between them.

Parameters

other	the pose to subtract.

operator/()

```
Pose2d Pose2d::operator/ (
            const double & scalar) const
```

Divides this pose by a scalar. Simply divides each component.

Parameters

scalar	the scalar value to divide by.

operator==()

```
bool Pose2d::operator== (
            const Pose2d other) const
```

Compares this to another pose.

Parameters

other	the other pose to compare to.

Returns

true if each of the components are within 1e-9 of each other.

relative_to()

```
Pose2d Pose2d::relative_to (
            const Pose2d & other) const
```

Finds the pose equivalent to this pose relative to another arbitrary pose rather than the origin.

Parameters

other	the pose representing the new origin.

Returns

this pose relative to another pose.

rotation()

```
Rotation2d Pose2d::rotation () const
```

Returns the rotational component.

Returns

the rotational component.

setRotationDeg()

```
void Pose2d::setRotationDeg (
            double rotDeg)
```

sets the ration value of the rotational component in Degrees

setRotationRad()

```
void Pose2d::setRotationRad (
            double rotRad)
```

sets the ration value of the rotational component in Radians

transform_by()

```
Pose2d Pose2d::transform_by (
            const Transform2d & transform) const
```

Adds a transform to this pose. Simply adds each component.

Parameters

transform	the change in pose.

Returns

the pose after being transformed.

translation()

```
Translation2d Pose2d::translation () const
```

Returns the translational component.

Returns

the translational component.

x()

```
double Pose2d::x () const
```

Returns the x value of the translational component.

Returns

the x value of the translational component.

y()

```
double Pose2d::y () const
```

Returns the y value of the translational component.

Returns

the y value of the translational component.

4.58.4 Friends And Related Symbol Documentation

operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const Pose2d & pose)  [friend]
```

Sends a pose to an output stream. Ex. std::cout << pose;

prints "Pose2d[x: (value), y: (value), rad: (radians), deg: (degrees)]"

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

- pose2d.h
- pose2d.cpp

4.59 PurePursuitCommand Class Reference

`#include <drive_commands.h>`

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

4.59.1 Detailed Description

Autocommand wrapper class for pure pursuit function in the TankDrive class

4.59.2 Constructor & Destructor Documentation

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)

4.59.3 Member Function Documentation

on_timeout()

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

Reset the drive system when it times out

run()

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

Direct call to TankDrive::pure_pursuit

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

- drive_commands.h
- drive_commands.cpp

4.60 Rect Struct Reference

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

4.60.1 Detailed Description

Describes a Rectangle with a minimum and maximum point

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

- geometry.h

4.61 robot_specs_t Struct Reference

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

Public Attributes

- double **robot_radius**

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

    the diameter of the wheels used for
- double **odom_gear_ratio**

    the ratio of the odometry wheel to the encoder reading odometry data
- double **dist_between_wheels**

    the distance between centers of the central drive wheels
- double drive_correction_cutoff
- 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

4.61.1 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.

4.61.2 Member Data Documentation

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

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

- robot_specs.h

4.62 Rotation2d Class Reference

```
#include <rotation2d.h>
```

Public Member Functions

- constexpr Rotation2d ()
- Rotation2d (const double &radians)
- Rotation2d (const double &x, const double &y)
- Rotation2d (const Translation2d &translation)
- double radians () const
- double degrees () const
- double revolutions () const
- double f_cos () const
- double f_sin () const
- double f_tan () const
- Eigen::Matrix2d rotation_matrix () const
- double wrapped_radians_180 () const
- double wrapped_degrees_180 () const
- double wrapped_revolutions_180 () const
- double wrapped_radians_360 () const
- double wrapped_degrees_360 () const
- double wrapped_revolutions_360 () const
- Rotation2d operator+ (const Rotation2d &other) const
- Rotation2d operator- (const Rotation2d &other) const
- Rotation2d operator- () const
- Rotation2d operator* (const double &scalar) const
- Rotation2d operator/ (const double &scalar) const
- bool operator== (const Rotation2d &other) const

Friends

- std::ostream & operator<< (std::ostream &os, const Rotation2d &rotation)

4.62.1 Detailed Description

Class representing a rotation in 2d space. Stores theta in radians, as well as cos and sin.

Internally this angle is stored continuously, however there are functions that return wrapped angles: "180" is from [-pi, pi), [-180, 180), [-0.5, 0.5) "360" is from [0, 2pi), [0, 360), [0, 1)

4.62.2 Constructor & Destructor Documentation

Rotation2d() [1/4]

```
Rotation2d::Rotation2d ()  [inline], [constexpr]
```

Default Constructor for Rotation2d

Rotation2d() [2/4]

```
Rotation2d::Rotation2d (
            const double & radians)
```

Constructs a rotation with the given value in radians.

Parameters

radians	the value of the rotation in radians.

Rotation2d() [3/4]

```
Rotation2d::Rotation2d (
            const double & x,
            const double & y)
```

Constructs a rotation given x and y values. Does not have to be normalized. The angle from the x axis to the point.

[theta] = [atan2(y, x)]

Parameters

x	the x value of the point
y	the y value of the point

Rotation2d() [4/4]

```
Rotation2d::Rotation2d (
            const Translation2d & translation)
```

Constructs a rotation given x and y values in the form of a Translation2d. Does not have to be normalized. The angle from the x axis to the point.

[theta] = [atan2(y, x)]

Parameters

translation	

4.62.3 Member Function Documentation

degrees()

```
double Rotation2d::degrees () const
```

Returns the degree angle value.

Returns

the degree angle value.

f_cos()

```
double Rotation2d::f_cos () const
```

Returns the cosine of the angle value.

Returns

the cosine of the angle value

f_sin()

```
double Rotation2d::f_sin () const
```

Returns the sine of the angle value.

Returns

the sine of the angle value.

f_tan()

```
double Rotation2d::f_tan () const
```

Returns the tangent of the angle value.

Returns

the tangent of the angle value.

operator*()

```
Rotation2d Rotation2d::operator* (
            const double & scalar) const
```

Multiplies this rotation by a scalar.

Parameters

scalar	the scalar value to multiply the rotation by.

Returns

the rotation multiplied by the scalar.

operator+()

```
Rotation2d Rotation2d::operator+ (
        const Rotation2d & other) const
```

Adds the values of two rotations using a rotation matrix

[new_cos] = [other.cos, -other.sin][cos] [new_sin] = [other.sin, other.cos][sin] new_value = atan2(new_sin, new_cos)

Parameters

other	the other rotation to add to this rotation.

Returns

the sum of the two rotations.

Adds the values of two rotations using a rotation matrix.

[new_cos] = [other.cos, -other.sin][cos] [new_sin] = [other.sin, other.cos][sin] new_value = atan2(new_sin, new_cos)

Parameters

other	the other rotation to add to this rotation.

Returns

the sum of the two rotations.

operator-() [1/2]

```
Rotation2d Rotation2d::operator- () const
```

Takes the inverse of this rotation by flipping it. Equivalent to adding 180 degrees.

Returns

this inverse of the rotation.

Takes the inverse of this rotation by flipping it.

Returns

this inverse of the rotation.

operator-() [2/2]

```
Rotation2d Rotation2d::operator- (
        const Rotation2d & other) const
```

Subtracts the values of two rotations.

Parameters

other	the other rotation to subtract from this rotation.

Returns

the difference between the two rotations.

operator/()

```
Rotation2d Rotation2d::operator/ (
            const double & scalar) const
```

Divides this rotation by a scalar.

Parameters

scalar	the scalar value to divide the rotation by.

Returns

the rotation divided by the scalar.

operator==()

```
bool Rotation2d::operator== (
            const Rotation2d & other) const
```

Compares two rotations. Returns true if their values are within 1e-9 radians of each other, to account for floating point error.

Parameters

other	the other rotation to compare to

Returns

whether the values of the rotations are within 1e-9 radians of each other

radians()

```
double Rotation2d::radians () const
```

Returns the radian angle value.

Returns

the radian angle value.

revolutions()

```
double Rotation2d::revolutions () const
```

Returns the revolution angle value.

Returns

the revolution angle value.

rotation_matrix()

```
Eigen::Matrix2d Rotation2d::rotation_matrix () const
```

Returns the rotation matrix equivalent to this rotation [cos, -sin] R = [sin, cos]

Returns

the rotation matrix equivalent to this rotation

wrapped_degrees_180()

```
double Rotation2d::wrapped_degrees_180 () const
```

Returns the degree angle value, wrapped from [-180, 180).

Returns

the degree angle value, wrapped from [-180, 180)

wrapped_degrees_360()

```
double Rotation2d::wrapped_degrees_360 () const
```

Returns the degree angle value, wrapped from [0, 360).

Returns

the degree angle value, wrapped from [0, 360)

wrapped_radians_180()

```
double Rotation2d::wrapped_radians_180 () const
```

Returns the radian angle value, wrapped from [-pi, pi).

Returns

the radian angle value, wrapped from [-pi, pi)

wrapped_radians_360()

```
double Rotation2d::wrapped_radians_360 () const
```

Returns the radian angle value, wrapped from [0, 2pi).

Returns

the radian angle value, wrapped from [0, 2pi)

wrapped_revolutions_180()

```
double Rotation2d::wrapped_revolutions_180 () const
```

Returns the revolution angle value, wrapped from [-0.5, 0.5).

Returns

the revolution angle value, wrapped from [-0.5, 0.5)

wrapped_revolutions_360()

```
double Rotation2d::wrapped_revolutions_360 () const
```

Returns the revolution angle value, wrapped from [0, 1).

Returns

the revolution angle value, wrapped from [0, 1)

4.62.4 Friends And Related Symbol Documentation

operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const Rotation2d & rotation)  [friend]
```

Sends a rotation to an output stream. Ex. std::cout << rotation;

prints "Rotation2d[rad: (radians), deg: (degrees)]"

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

- rotation2d.h
- rotation2d.cpp

4.63 ScaledSphericalSimplexSigmaPoints< STATES > Class Template Reference

```
#include <unscented_kalman_filter.h>
```

Public Member Functions

- ScaledSphericalSimplexSigmaPoints (double alpha=0.001, double beta=2)
- int num_sigmas ()
- EMat< STATES, NUM_SIGMAS > square_root_sigma_points (const EVec< STATES > &x, const EMat< STATES, STATES > &S)
- const EVec< NUM_SIGMAS > & Wm () const
- const EVec< NUM_SIGMAS > & Wc () const
- double Wm (int i) const
- double Wc (int i) const

4.63.1 Detailed Description

template<int STATES>
class ScaledSphericalSimplexSigmaPoints< STATES >

Generates sigma points and weights according to the paper [1] This is very different from Wan and Merwe's formulation.

This only requires N + 2 sigma points instead of 2N + 1 sigma points. Rather than generating sigma points symmetrically around the mean, it generates them as vertices of an N-simplex.

The performance of the filter using this reduced set of sigma points is identical to the standard method, so there is no downside to using it here.

[1] A Scaled Spherical Simplex Filter (S3F) with a decreased n + 2 sigma points set size and equivalent 2n + 1 Unscented Kalman Filter (UKF) accuracy

Template Parameters

STATES	the dimension of the state. NUM_SIGMAS sigma points and weights will be generated.

4.63.2 Constructor & Destructor Documentation

ScaledSphericalSimplexSigmaPoints()

```
template<int STATES>
ScaledSphericalSimplexSigmaPoints< STATES >::ScaledSphericalSimplexSigmaPoints (
            double alpha = 0.001,
            double beta = 2)  [inline]
```

Constructs a sigma point generator for Spherical Simplex sigma points

Parameters

alpha	Determines the spread of the sigma points around the mean. Smaller values are closer to the mean, this is usually a small value.
beta	Incorporates prior knowledge of the distribution of the state. For Gaussian distributions, beta = 2 is optimal.

4.63.3 Member Function Documentation

num_sigmas()

```
template<int STATES>
int ScaledSphericalSimplexSigmaPoints< STATES >::num_sigmas ()  [inline]
```

Returns the number of sigma points, for simplex sigma points this is N+2.

square_root_sigma_points()

```
template<int STATES>
EMat< STATES, NUM_SIGMAS > ScaledSphericalSimplexSigmaPoints< STATES >::square_root_sigma_←
points (
            const EVec< STATES > & x,
            const EMat< STATES, STATES > & S)  [inline]
```

Computes the sigma points given a mean (x) and square-root covariance (S).

Parameters

x	Vector of the means.
S	Square-root covariance.

Returns

Matrix containing the sigma points. Each column contains one sigma point in the same space as x. The first column is the same as the mean, with the others arranged around the mean.

Wc() [1/2]

```
template<int STATES>
const EVec< NUM_SIGMAS > & ScaledSphericalSimplexSigmaPoints< STATES >::Wc () const  [inline]
```

Returns a vector containing the weights of each sigma point for the covariance.

Wc() [2/2]

```
template<int STATES>
double ScaledSphericalSimplexSigmaPoints< STATES >::Wc (
            int i) const  [inline]
```

Returns the weight for the i-th sigma point for the covariance.

Parameters

i	Element of the weights vector to return.

Wm() [1/2]

```
template<int STATES>
const EVec< NUM_SIGMAS > & ScaledSphericalSimplexSigmaPoints< STATES >::Wm () const  [inline]
```

Returns a vector containing the weights of each sigma point for the mean.

Wm() [2/2]

```
template<int STATES>
double ScaledSphericalSimplexSigmaPoints< STATES >::Wm (
            int i) const  [inline]
```

Returns the weight for the i-th sigma point for the mean.

Parameters

i	Element of the weights vector to return.

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

- unscented_kalman_filter.h

4.64 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.

4.64.1 Detailed Description

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

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

- screen.cpp

4.65 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

4.65.1 Detailed Description

Serializes Arbitrary data to a file on the SD Card.

4.65.2 Constructor & Destructor Documentation

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

4.65.3 Member Function Documentation

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

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

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

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

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

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

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

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

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:

- serializer.h
- serializer.cpp

4.66 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

4.66.1 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.

4.66.2 Constructor & Destructor Documentation

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

4.66.3 Member Function Documentation

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:

- screen.h
- screen.cpp

4.67 SpinRPMCommand Class Reference

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

Public Member Functions

- SpinRPMCommand (Flywheel &flywheel, int rpm)
- bool run () override

4.67.1 Detailed Description

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

4.67.2 Constructor & Destructor Documentation

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]

4.67.3 Member Function Documentation

run()

```
bool SpinRPMCommand::run () [override]
```

Run spin_manual Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

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

- flywheel_commands.h
- flywheel_commands.cpp

4.68 PurePursuit::spline Struct Reference

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

4.68.1 Detailed Description

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

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

- pure_pursuit.h

4.69 StateMachine< System, IDType, Message, delay_ms, do_log >::State Struct Reference

`#include <state_machine.h>`

4.69.1 Detailed Description

template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log = false>
struct StateMachine< System, IDType, Message, delay_ms, do_log >::State

Abstract class that all states for this machine must inherit from States MUST override respond() and id() in order to function correctly (the compiler won't have it any other way)

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

- state_machine.h

4.70 StateMachine< System, IDType, Message, delay_ms, do_log > Class Template Reference

State Machine :)))))) A fun fun way of controlling stateful subsystems - used in the 2023-2024 Over Under game for our overly complex intake-cata subsystem (see there for an example) The statemachine runs in a background thread and a user thread can interact with it through current_state and send_message.

`#include <state_machine.h>`

Classes

- class MaybeMessage

  MaybeMessage a message of Message type or nothing MaybeMessage m = {}; // empty MaybeMessage m = Message::EnumField1.
- struct State

Public Member Functions

- StateMachine (State ∗initial)

  Construct a state machine and immediatly start running it.
- IDType current_state () const

  retrieve the current state of the state machine. This is safe to call from external threads
- void send_message (Message msg)

  send a message to the state machine from outside

4.70.1 Detailed Description

**template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log = false>
class StateMachine< System, IDType, Message, delay_ms, do_log >**

State Machine :)))))) A fun fun way of controlling stateful subsystems - used in the 2023-2024 Over Under game for our overly complex intake-cata subsystem (see there for an example) The statemachine runs in a background thread and a user thread can interact with it through current_state and send_message.

Designwise: the System class should hold onto any motors, feedback controllers, etc that are persistent in the system States themselves should hold any data that *only* that state needs. For example if a state should be exitted after a certain amount of time, it should hold a timer rather than the System holding that timer. (see Junder from 2024 for an example of this design)

Template Parameters

System	The system that this is the base class of `class Thing : public StateMachine<Thing> @tparam IDType The ID enum that recognizes states. Hint hint, use an`enum class`
Message	the message enum that a state or an outside can send and that states respond to
delay_ms	the delay to wait between each state processing to allow other threads to work
do_log	true if you want print statements describing incoming messages and current states. If true, it is expected that IDType and Message have a function called to_string that takes them as its only parameter and returns a std::string

4.70.2 Constructor & Destructor Documentation

StateMachine()

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log =
false>
StateMachine< System, IDType, Message, delay_ms, do_log >::StateMachine (
            State * initial) [inline]
```

Construct a state machine and immediatly start running it.

Parameters

initial	the state that the machine will begin in

4.70.3 Member Function Documentation

current_state()

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log =
false>
IDType StateMachine< System, IDType, Message, delay_ms, do_log >::current_state () const
[inline]
```

retrieve the current state of the state machine. This is safe to call from external threads

Returns

the current state

send_message()

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log =
false>
void StateMachine< System, IDType, Message, delay_ms, do_log >::send_message (
            Message msg)  [inline]
```

send a message to the state machine from outside

Parameters

msg	the message to send This is safe to call from external threads

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

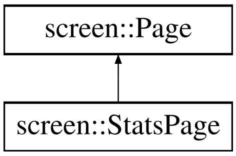
- state_machine.h

4.71 screen::StatsPage Class Reference

Draws motor stats and battery stats to the screen.

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

Inheritance diagram for screen::StatsPage:



Public Member Functions

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

    Creates a stats page.
- void update (bool was_pressed, int x, int y) override
- void draw (vex::brain::lcd &, bool first_draw, unsigned int frame_number) override

4.71.1 Detailed Description

Draws motor stats and battery stats to the screen.

4.71.2 Constructor & Destructor Documentation

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

4.71.3 Member Function Documentation

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.

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:

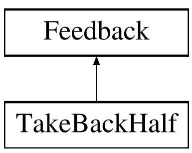
- screen.h
- screen.cpp

4.72 TakeBackHalf Class Reference

A velocity controller.

`#include <take_back_half.h>`

Inheritance diagram for TakeBackHalf:



Public Member Functions

- void init (double start_pt, double set_pt)
- 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

4.72.1 Detailed Description

A velocity controller.

Warning

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

4.72.2 Member Function Documentation

get()

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

Returns

the last saved result from the feedback controller

Implements Feedback.

init()

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

is_on_target()

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

Returns

true if the feedback controller has reached it's setpoint

Implements Feedback.

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.

update()

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

Iterate the feedback loop once with an updated sensor value

Parameters

val	value from the sensor

Returns

feedback loop result

Implements Feedback.

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

- take_back_half.h
- take_back_half.cpp

4.73 TankDrive Class Reference

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

Public Types

- enum class BrakeType { None , ZeroVelocity , Smart , **TurnOnly** }

Public Member Functions

- TankDrive (motor_group &left_motors, motor_group &right_motors, robot_specs_t &config, OdometryBase ∗odom=NULL)
- void stop ()
- Pose2d get_position ()
- 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)

Static Public Member Functions

- static double modify_inputs (double input, int power=2)

4.73.1 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

4.73.2 Member Enumeration Documentation

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

4.73.3 Constructor & Destructor Documentation

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
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

4.73.4 Member Function Documentation

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

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

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

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

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)

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
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

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

get_position()

```
Pose2d TankDrive::get_position ()
```

Returns the Robot position as a Pose2d

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

pure_pursuit()

```
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

reset_auto()

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

Reset the initialization for autonomous drive functions

stop()

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

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

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

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

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

turn_to_heading() [2/2]

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

Turn the robot in place to an exact heading relative to the field. 0 is forward.

Parameters

heading_deg	the heading to which we will turn
feedback	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Turn the robot in place to an exact heading relative to the field. 0 is forward.

Parameters

heading_deg	the heading to which we will turn
feedback	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have reached our target heading

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

- tank_drive.h
- tank_drive.cpp

4.74 tracking_wheel_cfg_t Struct Reference

```
#include <odometry_nwheel.h>
```

Public Attributes

- double x
- double y
- double theta_rad
- double radius

4.74.1 Detailed Description

OdometryNWheel

This class handles the code for an N-pod odometry setup, where there are N <WHEELS> free spinning omni wheels (dead wheels) placed in any known configuration on the robot.

Example of a possible wheel configuration:

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

Where O is the center of rotation. The robot will monitor the changes in rotation of these wheels, use this to calculate a pose delta, then integrate the deltas over time to determine the robot's position.

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.

https://rit.enterprise.slack.com/files/U04112Y5RB6/F080M01KPA5/predictperpindiculars2.pdf 2024-2025 Notebook: Entries/Software Entries/Localization/N-Pod Odometry

Author

Jack Cammarata, Richie Sommers

Date

Nov 14 2024 tracking_wheel_cfg_t holds all the specifications for a single tracking wheel The units for x, y, and radius will determine the units of the position estimate

4.74.2 Member Data Documentation

radius

```
double tracking_wheel_cfg_t::radius
```

radius of the wheel

theta_rad

```
double tracking_wheel_cfg_t::theta_rad
```

angle between wheel direction and x axis in the robot frame

x

```
double tracking_wheel_cfg_t::x
```

x position of the center of the wheel

y

```
double tracking_wheel_cfg_t::y
```

y position of the center of the wheel

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

- odometry_nwheel.h

4.75 Transform2d Class Reference

```
#include <transform2d.h>
```

Public Member Functions

- constexpr Transform2d ()
- Transform2d (const Translation2d &translation, const Rotation2d &rotation)
- Transform2d (const double &x, const double &y, const Rotation2d &rotation)
- Transform2d (const double &x, const double &y, const double &radians)
- Transform2d (const Translation2d &translation, const double &radians)
- Transform2d (const Eigen::Vector3d &transform_vector)
- Transform2d (const Pose2d &start, const Pose2d &end)
- Translation2d translation () const
- double x () const
- double y () const
- Rotation2d rotation () const
- Transform2d inverse () const
- Transform2d operator* (const double &scalar) const
- Transform2d operator/ (const double &scalar) const
- Transform2d operator- () const
- bool operator== (const Transform2d &other) const

Friends

- std::ostream & operator<< (std::ostream &os, const Transform2d &transform)

4.75.1 Detailed Description

Class representing a transformation of a pose2d, or a linear difference between the components of poses.

Assumes conventional cartesian coordinate system: Looking down at the coordinate plane, +X is right +Y is up +Theta is counterclockwise

4.75.2 Constructor & Destructor Documentation

Transform2d() [1/7]

```
Transform2d::Transform2d ()  [constexpr]
```

Default Constructor for Transform2d

Transform2d() [2/7]

```
Transform2d::Transform2d (
            const Translation2d & translation,
            const Rotation2d & rotation)
```

Constructs a transform given translation and rotation components.

Parameters

translation	the translational component of the transform.
rotation	the rotational component of the transform.

Transform2d() [3/7]

```
Transform2d::Transform2d (
            const double & x,
            const double & y,
            const Rotation2d & rotation)
```

Constructs a transform given translation and rotation components.

Parameters

x	the x component of the transform.
y	the y component of the transform.
rotation	the rotational component of the transform.

Transform2d() [4/7]

```
Transform2d::Transform2d (
            const double & x,
            const double & y,
            const double & radians)
```

Constructs a transform given translation and rotation components.

Parameters

x	the x component of the transform.
y	the y component of the transform.
radians	the rotational component of the transform in radians.

Transform2d() [5/7]

```
Transform2d::Transform2d (
            const Translation2d & translation,
            const double & radians)
```

Constructs a transform given translation and rotation components.

Parameters

translation	the translational component of the transform.
radians	the rotational component of the transform in radians.

Transform2d() [6/7]

```
Transform2d::Transform2d (
            const Eigen::Vector3d & transform_vector)
```

Constructs a transform given translation and rotation components given as a vector.

Parameters

transform_vector	vector of the form [x, y, theta]

Transform2d() [7/7]

```
Transform2d::Transform2d (
            const Pose2d & start,
            const Pose2d & end)
```

Constructs a transform given translation and rotation components.

Parameters

translation	the translational component of the transform.
rotation	the rotational component of the transform.

4.75.3 Member Function Documentation

inverse()

```
Transform2d Transform2d::inverse () const
```

Inverts the transform.

Returns

the inverted transform.

operator*()

```
Transform2d Transform2d::operator* (
             const double & scalar) const
```

Multiplies this transform by a scalar.

Parameters

scalar	the scalar to multiply this transform by.

operator-()

```
Transform2d Transform2d::operator- () const
```

Inverts the transform.

Returns

the inverted transform.

operator/()

```
Transform2d Transform2d::operator/ (
             const double & scalar) const
```

Divides this transform by a scalar.

Parameters

scalar	the scalar to divide this transform by.

operator==()

```
bool Transform2d::operator== (
            const Transform2d & other) const
```

Compares this to another transform.

Parameters

other	the other transform to compare to.

Returns

true if the components are within 1e-9 of each other.

rotation()

```
Rotation2d Transform2d::rotation () const
```

Returns the rotational component of the transform.

Returns

the rotational component of the transform.

translation()

```
Translation2d Transform2d::translation () const
```

Returns the translational component of the transform.

Returns

the translational component of the transform.

x()

```
double Transform2d::x () const
```

Returns the x component of the transform.

Returns

the x component of the transform.

y()

```
double Transform2d::y () const
```

Returns the y component of the transform.

Returns

the y component of the transform.

4.75.4 Friends And Related Symbol Documentation

operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const Transform2d & transform)  [friend]
```

Sends a transform to an output stream. Ex. std::cout << transform;

prints "Transform2d[dx: (value), dy: (value), drad: (radians), ddeg: (degrees)]"

Sends a transform to an output stream. Ex. std::cout << transform;

prints "Transform2d[x: (value), y: (value), rad: (radians), deg: (degrees)]"

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

- transform2d.h
- transform2d.cpp

4.76 Translation2d Class Reference

```
#include <translation2d.h>
```

Public Member Functions

- constexpr Translation2d ()
- Translation2d (const double &x, const double &y)
- Translation2d (const Eigen::Vector2d &vector)
- Translation2d (const double &r, const Rotation2d &theta)
- double x () const
- void setX (double x)
- double y () const
- void setY (double y)
- Rotation2d theta () const
- Eigen::Vector2d as_vector () const
- double norm () const
- Translation2d normalize () const
- double distance (const Translation2d &other) const
- Translation2d rotate_by (const Rotation2d &rotation) const
- Translation2d rotate_around (const Translation2d &other, const Rotation2d &rotation) const
- Translation2d operator+ (const Translation2d &other) const
- Translation2d operator- (const Translation2d &other) const
- Translation2d operator- () const
- Translation2d operator* (const double &scalar) const
- Translation2d operator/ (const double &scalar) const
- double operator* (const Translation2d &other) const
- bool operator== (const Translation2d &other) const

Friends

- std::ostream & operator<< (std::ostream &os, const Translation2d &translation)

4.76.1 Detailed Description

Class representing a point in 2d space with x and y coordinates.

Assumes conventional cartesian coordinate system: Looking down at the coordinate plane, +X is right +Y is up +Theta is counterclockwise

4.76.2 Constructor & Destructor Documentation

Translation2d() [1/4]

```
Translation2d::Translation2d ()  [inline], [constexpr]
```

Default Constructor for Translation2d

Translation2d() [2/4]

```
Translation2d::Translation2d (
              const double & x,
              const double & y)
```

Constructs a Translation2d with the given x and y values.

Parameters

x	The x component of the translation.
y	The y component of the translation.

Translation2d() [3/4]

```
Translation2d::Translation2d (
              const Eigen::Vector2d & vector)
```

Constructs a Translation2d with the values from the given vector.

Parameters

vector	The vector whose values will be used.

Translation2d() [4/4]

```
Translation2d::Translation2d (
              const double & r,
              const Rotation2d & theta)
```

Constructs a Translation2d given polar coordinates of the form (r, theta).

Parameters

r	The radius (magnitude) of the vector.
theta	The angle (direction) of the vector.

4.76.3 Member Function Documentation

as_vector()

```
Eigen::Vector2d Translation2d::as_vector () const
```

Returns the vector as an Eigen::Vector2d.

Returns

Eigen::Vector2d with the same values as the translation.

distance()

```
double Translation2d::distance (
            const Translation2d & other) const
```

Returns the distance between two translations.

Returns

the distance between two translations.

norm()

```
double Translation2d::norm () const
```

Returns the norm/radius/magnitude/distance from origin.

Returns

the norm of the translation.

normalize()

```
Translation2d Translation2d::normalize () const
```

returns a translation as if it were a vector with a magnitude of 1

Returns

the norm of the translation.

Returns a translation so that it has a vector magnitude of 1

Returns

the norm of the translation.

operator*() [1/2]

```
Translation2d Translation2d::operator* (
        const double & scalar) const
```

Returns this translation multiplied by a scalar.

[x] = [x] * [scalar] [y] = [y] * [scalar]

Parameters

scalar	the scalar to multiply by.

Returns

this translation multiplied by a scalar.

operator*() [2/2]

```
double Translation2d::operator* (
            const Translation2d & other) const
```

Returns the dot product of two translations.

[scalar] = [x][otherx] + [y][othery]

Parameters

other	the other translation to find the dot product with.

Returns

the scalar valued dot product.

operator+()

```
Translation2d Translation2d::operator+ (
            const Translation2d & other) const
```

Returns the sum of two translations.

[x] = [x] + [otherx]; [y] = [y] + [othery];

Parameters

other	the other translation to add to this translation.

Returns

the sum of the two translations.

operator-() [1/2]

```
Translation2d Translation2d::operator- () const
```

Returns the inverse of this translation. Equivalent to flipping the vector across the origin.

[x] = [-x] [y] = [-y]

Returns

the inverse of this translation.

operator-() [2/2]

```
Translation2d Translation2d::operator- (
            const Translation2d & other) const
```

Returns the difference of two translations.

[x] = [x] - [otherx] [y] = [y] - [othery]

Parameters

other	the translation to subtract from this translation.

Returns

the difference of the two translations.

operator/()

```
Translation2d Translation2d::operator/ (
            const double & scalar) const
```

Returns this translation divided by a scalar.

[x] = [x] / [scalar] [y] = [y] / [scalar]

Parameters

scalar	the scalar to divide by.

Returns

this translation divided by a scalar.

operator==()

```
bool Translation2d::operator== (
            const Translation2d & other) const
```

Compares two translations. Returns true if their components are each within 1e-9, to account for floating point error.

Parameters

other	the translation to compare to.

Returns

whether the two translations are equal.

rotate_around()

```
Translation2d Translation2d::rotate_around (
            const Translation2d & other,
            const Rotation2d & rotation) const
```

Applies a rotation to this translation around another given point.

[x] = [cos, -sin][x - otherx] + [otherx] [y] = [sin, cos][y - othery] + [othery]

Parameters

other	the center of rotation.
rotation	the angle amount the translation will be rotated.

Returns

the translation that has been rotated.

rotate_by()

```
Translation2d Translation2d::rotate_by (
            const Rotation2d & rotation) const
```

Applies a rotation to this translation around the origin.

Equivalent to multiplying a vector by a rotation matrix: x = [cos, -sin][x] y = [sin, cos][y]

Parameters

rotation	the angle amount the translation will be rotated.

Returns

the new translation that has been rotated around the origin.

setX()

```
void Translation2d::setX (
            double x)
```

Sets the x value of the translation.

setY()

```
void Translation2d::setY (
            double y)
```

Sets the y value of the translation.

theta()

```
Rotation2d Translation2d::theta () const
```

Returns the angle of the translation.

Returns

the angle of the translation.

x()

```
double Translation2d::x () const
```

Returns the x value of the translation.

Returns

the x value of the translation.

y()

```
double Translation2d::y () const
```

Returns the y value of the translation.

Returns

the y value of the translation.

4.76.4 Friends And Related Symbol Documentation

operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const Translation2d & translation)  [friend]
```

Sends a translation to an output stream. Ex. std::cout << translation;

prints "Translation2d[x: (value), y: (value)]"

Sends a translation to an output stream. Ex. std::cout << translation;

prints "Translation2d[x: (value), y: (value), rad: (radians), deg: (degrees)]"

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

- translation2d.h
- translation2d.cpp

4.77 TrapezoidProfile Class Reference

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

Public Member Functions

- TrapezoidProfile (const double &x_initial, const double &x_target, const double &v_max, const double &accel, const double &decel)
- motion_t calculate (double t)
- double total_time ()

4.77.1 Detailed Description

Class representing a trapezoidal motion profile. This consists of either two or three stages. First, an acceleration stage where the velocity increases to a maximum. Then a cruise stage where the velocity is constant, then a deceleration stage where the velocity decreases to zero.

This implementation allows for different acceleration and deceleration rates.

This is best used with LQR, as it tracks both velocity and position at the same time, however it can also be used with PID and feedforward.

Author

Jack Cammarata

Date

3/28/2025

4.77.2 Constructor & Destructor Documentation

TrapezoidProfile()

```
TrapezoidProfile::TrapezoidProfile (
            const double & x_initial,
            const double & x_target,
            const double & v_max,
            const double & accel,
            const double & decel)
```

Constructs a TrapezoidProfile.

Parameters

x_initial	The initial position.
x_target	The target position.
v_max	The maximum velocity.
accel	The acceleration.
decel	The deceleration.

4.77.3 Member Function Documentation

calculate()

```
motion_t TrapezoidProfile::calculate (
            double t)
```

Calculate the state along the motion profile at some given time.

Parameters

t	The time in seconds.

Returns

the state.

Calculate the state along the motion profile after some given time.

Parameters

t	The time in seconds.

Returns

the state.

total_time()

```
double TrapezoidProfile::total_time ()
```

Returns the total time that the motion profile takes to complete.

Returns

the total time that the motion profile takes to complete.

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

- trapezoid_profile.h
- trapezoid_profile.cpp

4.78 TurnDegreesCommand Class Reference

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

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

4.78.1 Detailed Description

AutoCommand wrapper class for the turn_degrees function in the TankDrive class

4.78.2 Constructor & Destructor Documentation

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

4.78.3 Member Function Documentation

on_timeout()

```
void TurnDegreesCommand::on_timeout ()  [override]
```

Cleans up drive system if we time out before finishing

reset the drive system if we timeout

run()

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

Run turn_degrees Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

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

- drive_commands.h
- drive_commands.cpp

4.79 TurnToHeadingCommand Class Reference

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

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

4.79.1 Detailed Description

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

4.79.2 Constructor & Destructor Documentation

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

4.79.3 Member Function Documentation

on_timeout()

```
void TurnToHeadingCommand::on_timeout () [override]
```

Cleans up drive system if we time out before finishing

reset the drive system if we don't hit our target

run()

```
bool TurnToHeadingCommand::run () [override]
```

Run turn_to_heading Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

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

- drive_commands.h
- drive_commands.cpp

4.80 Twist2d Class Reference

```
#include <twist2d.h>
```

Public Member Functions

- constexpr Twist2d ()
- Twist2d (const double &dx, const double &dy, const double &dtheta)
- Twist2d (const Eigen::Vector3d &twist_vector)
- double dx () const
- double dy () const
- double dtheta () const
- bool operator== (const Twist2d &other) const
- Twist2d operator* (const double &scalar) const
- Twist2d operator/ (const double &scalar) const

Friends

- std::ostream & operator<< (std::ostream &os, const Twist2d &twist)

4.80.1 Detailed Description

Class representing a difference between two poses, more specifically a distance along an arc from a pose.

Assumes conventional cartesian coordinate system: Looking down at the coordinate plane, +X is right +Y is up +Theta is counterclockwise

4.80.2 Constructor & Destructor Documentation

Twist2d() [1/3]

```
Twist2d::Twist2d ()  [constexpr]
```

Default Constructor for Twist2d

Twist2d() [2/3]

```
Twist2d::Twist2d (
            const double & dx,
            const double & dy,
            const double & dtheta)
```

Constructs a twist with given translation and angle deltas.

Parameters

dx	the linear dx component.
dy	the linear dy component.
dtheta	the angular dtheta component.

Twist2d() [3/3]

```
Twist2d::Twist2d (
            const Eigen::Vector3d & twist_vector)
```

Constructs a twist with given translation and angle deltas.

Parameters

twist_vector	vector of the form [dx, dy, dtheta]

4.80.3 Member Function Documentation

dtheta()

```
double Twist2d::dtheta () const
```

Returns the angular dtheta component.

Returns

the angular dtheta component.

dx()

```
double Twist2d::dx () const
```

Returns the linear dx component.

Returns

the linear dx component.

dy()

```
double Twist2d::dy () const
```

Returns the linear dy component.

Returns

the linear dy component.

operator*()

```
Twist2d Twist2d::operator* (
            const double & scalar) const
```

Multiplies this twist by a scalar.

Parameters

scalar	the scalar value to multiply by.

operator/()

```
Twist2d Twist2d::operator/ (
            const double & scalar) const
```

Divides this twist by a scalar.

Parameters

scalar	the scalar value to divide by.

operator==()

```
bool Twist2d::operator== (
            const Twist2d & other) const
```

Compares this to another twist.

Parameters

other	the other twist to compare to.

Returns

true if each of the components are within 1e-9 of each other.

4.80.4 Friends And Related Symbol Documentation

operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const Twist2d & twist) [friend]
```

Sends a twist to an output stream. Ex. std::cout << twist;

prints "Twist2d[dx: (value), dy: (value), drad: (radians)]"

Sends a twist to an output stream. Ex. std::cout << twist;

prints "Twist2d[x: (value), y: (value), rad: (radians), deg: (degrees)]"

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

- twist2d.h
- twist2d.cpp

4.81 UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS > Class Template Reference

```
#include <unscented_kalman_filter.h>
```

Public Member Functions

- UnscentedKalmanFilter (const std::function< StateVector(const StateVector &, const InputVector &)> &f, const std::function< OutputVector(const StateVector &, const InputVector &)> &h, const WithInputIntegrator &integrator, const StateVector &state_stddevs, const OutputVector &measurement_stddevs)
- UnscentedKalmanFilter (const std::function< StateVector(const StateVector &, const InputVector &)> &f, const std::function< OutputVector(const StateVector &, const InputVector &)> &h, const WithInputIntegrator &integrator, const StateVector &state_stddevs, const OutputVector &measurement_stddevs, const std::function< StateVector(const EMat< STATES, NUM_SIGMAS > &, const EVec< NUM_SIGMAS > &)> &mean_func_X, const std::function< OutputVector(const EMat< OUTPUTS, NUM_SIGMAS > &, const EVec< NUM_SIGMAS > &)> &mean_func_Y, const std::function< StateVector(const StateVector &, const StateVector &)> &residual_func_X, const std::function< OutputVector(const OutputVector &, const OutputVector &)> &residual_func_Y, const std::function< StateVector(const StateVector &, const StateVector &)> &add_func_X)
- StateMatrix S () const
- double S (int i, int j) const
- void set_S (const StateMatrix &S)
- StateMatrix P () const
- void set_P (const StateMatrix &P)
- StateVector xhat () const
- double xhat (int i) const
- void set_xhat (const StateVector &xhat)
- void set_xhat (int i, double value)
- void reset ()
- void predict (const InputVector &u, double dt)
- void correct (const InputVector &u, const OutputVector &y)
- void correct (const InputVector &u, const OutputVector &y, const EVec< OUTPUTS > &measurement_stddevs)
- template<int ROWS>
  void correct (const InputVector &u, const EVec< ROWS > &y, const std::function< EVec< ROWS >(const StateVector &, const InputVector &)> &h, const EVec< ROWS > &measurement_stddevs)
- template<int ROWS>
  void correct (const InputVector &u, const EVec< ROWS > &y, const std::function< EVec< ROWS >(const StateVector &, const InputVector &)> &h, const EVec< ROWS > measurement_stddevs, const std::function< EVec< ROWS >(const EMat< ROWS, NUM_SIGMAS > &, const EVec< NUM_SIGMAS > &)> &mean_func_Y, const std::function< EVec< ROWS >(const EVec< ROWS > &, const EVec< ROWS > &)> &residual_func_Y, const std::function< StateVector(const StateVector &, const StateVector &)> &residual_func_X, const std::function< StateVector(const StateVector &, const StateVector &)> &add_func_X)

4.81.1 Detailed Description

template<int STATES, int INPUTS, int OUTPUTS>
class UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >

Kalman filters combine predictions from a model and measurements to estimate a system's true state.

The Unscented Kalman Filter is a nonlinear estimator, meaning that the model used to predict how the state changes over time can be nonlinear. The model that determines the expected measurement given the current state can also be nonlinear.

At each timestep, sigma points are generated close to the mean, they are all propagated forward in time according to the nonlinear model. The Unscented Transform uses the propagated sigma points to compute the prior state and covariance.

When correcting the state and covariance with a measurement, sigma points are again generated, but are transformed into the measurement space using the measurement function. A Kalman gain matrix K is then computed, and used to update the state and covariance.

To read more about Kalman filters and the standard UKF read: https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python

This implementation is somewhat non-standard. The square-root form of the UKF (SR-UKF) is used, and the way the sigma points are generated is different than most implementations. The square-root form is used to ensure that the covariance matrix remains positive definite.

To learn more about the SR-UKF, and see the exact formulation that most of this implementation follows, read: https://www.researchgate.net/publication/3908304

The sigma points are not generated symmetrically around the mean, instead they are generated as vertices of a simplex. Using N = # of states, this method uses N + 2 sigma points instead of the standard 2N + 1 sigma points. This reduces computation up to 50%. To learn more about this method, read: https://www.sciencedirect.com/science/article/pii/S0888327020308190

This filter uses a method of "recalibrating" by essentially applying a measurement twice instead of once, and only using it if it is more accurate than before the measurement was applied. To learn more about this framework for nonlinear filters, read: https://arxiv.org/pdf/2407.05717

Template Parameters

STATES	Dimension of the state vector.
INPUTS	Dimension of the control input vector.
OUTPUTS	Dimension of the measurement vector.

4.81.2 Constructor & Destructor Documentation

UnscentedKalmanFilter() [1/2]

```
template<int STATES, int INPUTS, int OUTPUTS>
UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::UnscentedKalmanFilter (
            const std::function< StateVector(const StateVector &, const InputVector &)> & f,
            const std::function< OutputVector(const StateVector &, const InputVector &)> & h,
            const WithInputIntegrator & integrator,
            const StateVector & state_stddevs,
            const OutputVector & measurement_stddevs)  [inline]
```

Constructs an Unscented Kalman Filter.

Parameters

f	A vector valued function of x and u that returns the derivative of the state vector with respect to time.
h	A vector valued function of x and u that returns the expected measurement at the given state.
integrator	A function from "numerical_integration.h" that integrates a differential equation of the form f(x, u).
state_stddevs	Standard deviations of the states in the model.
measurement_stddevs	Standard deviations of the measurements.

UnscentedKalmanFilter() [2/2]

```
template<int STATES, int INPUTS, int OUTPUTS>
UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::UnscentedKalmanFilter (
            const std::function< StateVector(const StateVector &, const InputVector &)> & f,
            const std::function< OutputVector(const StateVector &, const InputVector &)> & h,
            const WithInputIntegrator & integrator,
            const StateVector & state_stddevs,
            const OutputVector & measurement_stddevs,
            const std::function< StateVector(const EMat< STATES, NUM_SIGMAS > &, const EVec<
NUM_SIGMAS > &)> & mean_func_X,
            const std::function< OutputVector(const EMat< OUTPUTS, NUM_SIGMAS > &, const
EVec< NUM_SIGMAS > &)> & mean_func_Y,
            const std::function< StateVector(const StateVector &, const StateVector &)> &
residual_func_X,
            const std::function< OutputVector(const OutputVector &, const OutputVector &)> &
residual_func_Y,
            const std::function< StateVector(const StateVector &, const StateVector &)> &
add_func_X)  [inline]
```

Constructs an Unscented Kalman Filter with custom mean, residual, and addition functions. The most common use for these functions is when you are estimating angles whose arithmetic operations need to be wrapped.

Parameters

f	A vector valued function of x and u that returns the derivative of the state vector with respect to time.
h	A vector valued function of x and u that returns the expected measurement at the given state.
integrator	A function from "numerical_integration.h" that integrates a differential equation of the form f(x, u).
state_stddevs	Standard deviations of the states in the model.
measurement_stddevs	Standard deviations of the measurements.
mean_func_X	A function that computes the mean of a matrix containing NUM_SIGMAS state sigma points with a set of weights for each.
mean_func_Y	A function that computes the mean of a matrix containing NUM_SIGMAS measurement sigma points with a set of weights for each.
residual_func_X	A function that computes the residual of two state vectors, usually by simple subtraction.
residual_func_Y	A function that computes the residual of two measurement vectors, usually by simple subtraction.
add_funx_X	A function that adds two state vectors.

4.81.3 Member Function Documentation

correct() [1/4]

```
template<int STATES, int INPUTS, int OUTPUTS>
template<int ROWS>
void UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::correct (
            const InputVector & u,
            const EVec< ROWS > & y,
            const std::function< EVec< ROWS >(const StateVector &, const InputVector &)> &
```

```
h,
            const EVec< ROWS > & measurement_stddevs)  [inline]
```

Correct the state estimate using the measurements in y, a custom measurement function, and custom standard deviations. This is useful for when a different set of measurements are being applied.

Parameters

u	The control input used in the last predict step.
y	The vector of measurements.
h	A vector valued function of x and u that returns the expected measurement at the given state.
measurement_stddevs	The vector of standard deviations for each measurement to be used for this correct step.

correct() [2/4]

```
template<int STATES, int INPUTS, int OUTPUTS>
template<int ROWS>
void UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::correct (
            const InputVector & u,
            const EVec< ROWS > & y,
            const std::function< EVec< ROWS >(const StateVector &, const InputVector &)> &
h,
            const EVec< ROWS > measurement_stddevs,
            const std::function< EVec< ROWS >(const EMat< ROWS, NUM_SIGMAS > &, const EVec<
NUM_SIGMAS > &)> & mean_func_Y,
            const std::function< EVec< ROWS >(const EVec< ROWS > &, const EVec< ROWS > &)>
& residual_func_Y,
            const std::function< StateVector(const StateVector &, const StateVector &)> &
residual_func_X,
            const std::function< StateVector(const StateVector &, const StateVector &)> &
add_func_X)  [inline]
```

Correct the state estimate using the measurements in y, a custom measurement function, custom standard deviations, and custom mean, residual, and addition functions. This is useful for when a different set of measurements are being applied, and they require custom arithmetic functions.

Parameters

u	The control input used in the last predict step.
y	The vector of measurements.
h	A vector valued function of x and u that returns the expected measurement at the given state.
measurement_stddevs	The vector of standard deviations for each measurement to be used for this correct step.
mean_func_Y	A function that computes the mean of a matrix containing NUM_SIGMAS measurement sigma points with a set of weights for each.
residual_func_X	A function that computes the residual of two state vectors, usually by simple subtraction.
residual_func_Y	A function that computes the residual of two measurement vectors, usually by simple subtraction.
add_funx_X	A function that adds two state vectors.

correct() [3/4]

```
template<int STATES, int INPUTS, int OUTPUTS>
void UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::correct (
            const InputVector & u,
            const OutputVector & y)  [inline]
```

Correct the state estimate using the measurements in y.

Parameters

u	The control input used in the last predict step.
y	The vector of measurements.

correct() [4/4]

```
template<int STATES, int INPUTS, int OUTPUTS>
void UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::correct (
            const InputVector & u,
            const OutputVector & y,
            const EVec< OUTPUTS > & measurement_stddevs)  [inline]
```

Correct the state estimate using the measurements in y, and custom standard deviations. This is useful for when the noise in the measurements vary.

Parameters

u	The control input used in the last predict step.
y	The vector of measurements.
measurement_stddevs	The vector of standard deviations for each measurement to be used for this correct step.

P()

```
template<int STATES, int INPUTS, int OUTPUTS>
StateMatrix UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::P () const  [inline]
```

Returns the reconstructed covariance matrix P.

predict()

```
template<int STATES, int INPUTS, int OUTPUTS>
void UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::predict (
            const InputVector & u,
            double dt)  [inline]
```

Projects the state into the future by dt seconds with control input u.

Parameters

u	The control input.
dt	The timestep in seconds.

reset()

```
template<int STATES, int INPUTS, int OUTPUTS>
void UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::reset ()  [inline]
```

Resets the filter.

S() [1/2]

```
template<int STATES, int INPUTS, int OUTPUTS>
StateMatrix UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::S () const  [inline]
```

Returns the square-root covariance matrix S.

S() [2/2]

```
template<int STATES, int INPUTS, int OUTPUTS>
double UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::S (
            int i,
            int j) const  [inline]
```

Returns one element of the square-root covariance matrix S.

Parameters

i	Row of S.
j	Column of S.

set_P()

```
template<int STATES, int INPUTS, int OUTPUTS>
void UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::set_P (
            const StateMatrix & P)  [inline]
```

Set the current square-root covariance matrix S to the square-root of P.

Parameters

P	The covariance matrix P.

set_S()

```
template<int STATES, int INPUTS, int OUTPUTS>
void UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::set_S (
            const StateMatrix & S)  [inline]
```

Set the current square-root covariance matrix S.

Parameters

S	The new square-root covariance matrix S.

set_xhat() [1/2]

```
template<int STATES, int INPUTS, int OUTPUTS>
void UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::set_xhat (
             const StateVector & xhat)  [inline]
```

Set the current state estimate x-hat.

set_xhat() [2/2]

```
template<int STATES, int INPUTS, int OUTPUTS>
void UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::set_xhat (
             int i,
             double value)  [inline]
```

Set one element of the current state estimate x-hat.

Parameters

i	Row of x-hat.

xhat() [1/2]

```
template<int STATES, int INPUTS, int OUTPUTS>
StateVector UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::xhat () const  [inline]
```

Returns the current state estimate x-hat.

xhat() [2/2]

```
template<int STATES, int INPUTS, int OUTPUTS>
double UnscentedKalmanFilter< STATES, INPUTS, OUTPUTS >::xhat (
             int i) const  [inline]
```

Returns one element of the current state estimate x-hat.

Parameters

i	Row of x-hat.

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

- unscented_kalman_filter.h

4.82 WaitUntilCondition Class Reference

Waits until the condition is true.

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

4.82.1 Detailed Description

Waits until the condition is true.

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

- auto_command.h

4.83 WaitUntilUpToSpeedCommand Class Reference

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

Public Member Functions

- WaitUntilUpToSpeedCommand (Flywheel &flywheel, int threshold_rpm)
- bool run () override

4.83.1 Detailed Description

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

4.83.2 Constructor & Destructor Documentation

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

4.83.3 Member Function Documentation

run()

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

Run spin_manual Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

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

- flywheel_commands.h
- flywheel_commands.cpp

Index