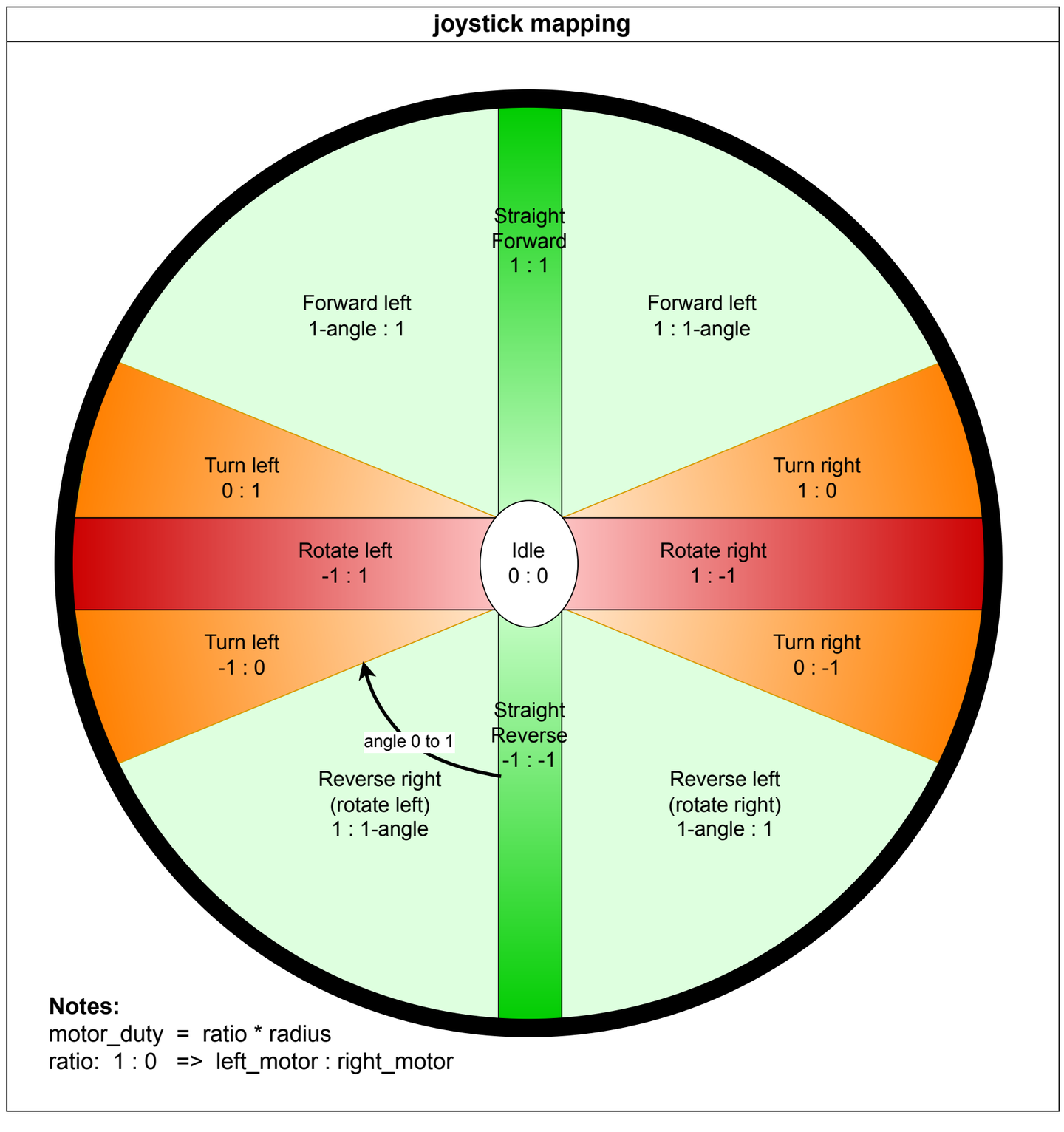


joystick mapping
Straight
Forward
1 : 1
Forward left
1-angle : 1
Forward left
1 : 1-angle
Turn left
0 : 1
Turn right
1 : 0
Rotate left
-1 : 1
Idle
0 : 0
Rotate right
1 : -1
Turn left
-1 : 0
Turn right
0 : -1
angle 0 to 1
Straight
Reverse
-1 : -1
Reverse right
(rotate left)
1 : 1-angle
Reverse left
(rotate right)
1-angle : 1
Notes:
motor_duty = ratio * radius
ratio: 1 : 0 => left_motor : right_motor

UART board communication (V2.1)

Planning data / structs need to be sent between the two boards here:

MOTORCTL_STATUS
[interval, slow]

timestamp
motorcommands (current actual duty, direction)
motor currents
axle rotating speed
temperatures
- motors
- drivers
- brake resistors

fan target state [ ] handled by motorctl board
battery voltage
config?

board control

- joystick
- encoder, switch
- display

UART

board motorctl

- 2x driver
- 2x currentsensor
- 2x light barrier axle speed
- 2x brake resistor
- temp sensors
  - motor
  - driver
  - brake resistor
- fan

MOTOR COMMANDS
[interval, fast]

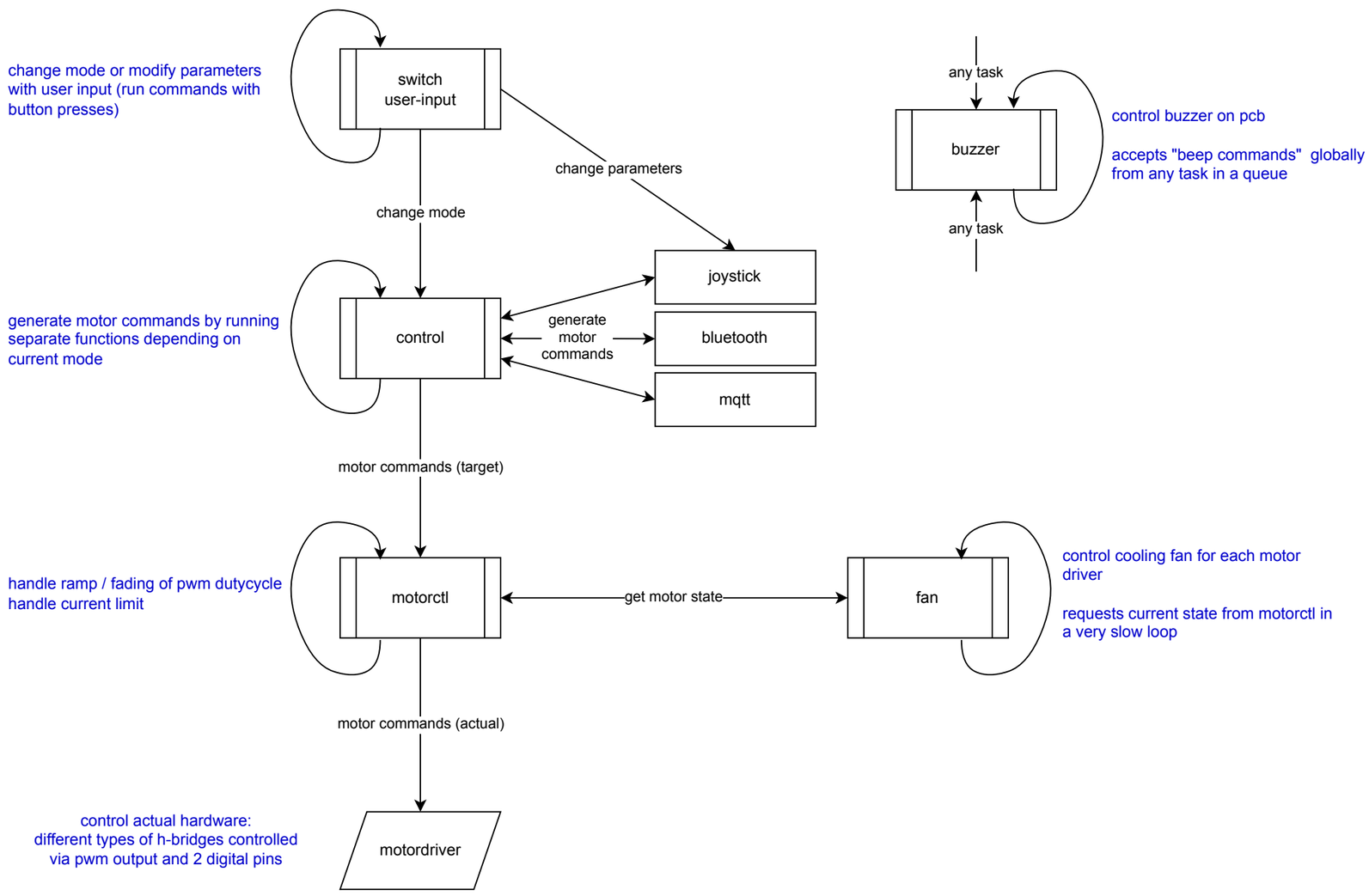
timestamp
motorcommands
- motorCommand_t left
  - motorstate
  - duty

CONFIG
[manual / on change]

timestamp
motorctl_config_t
- msFadeAccel
- msFadeDecel
- bool currentLimitEnabled
- (currentSensor_adc)
- (currentSensor_ratedCurrent)
- currentMax
- deadTimeMs

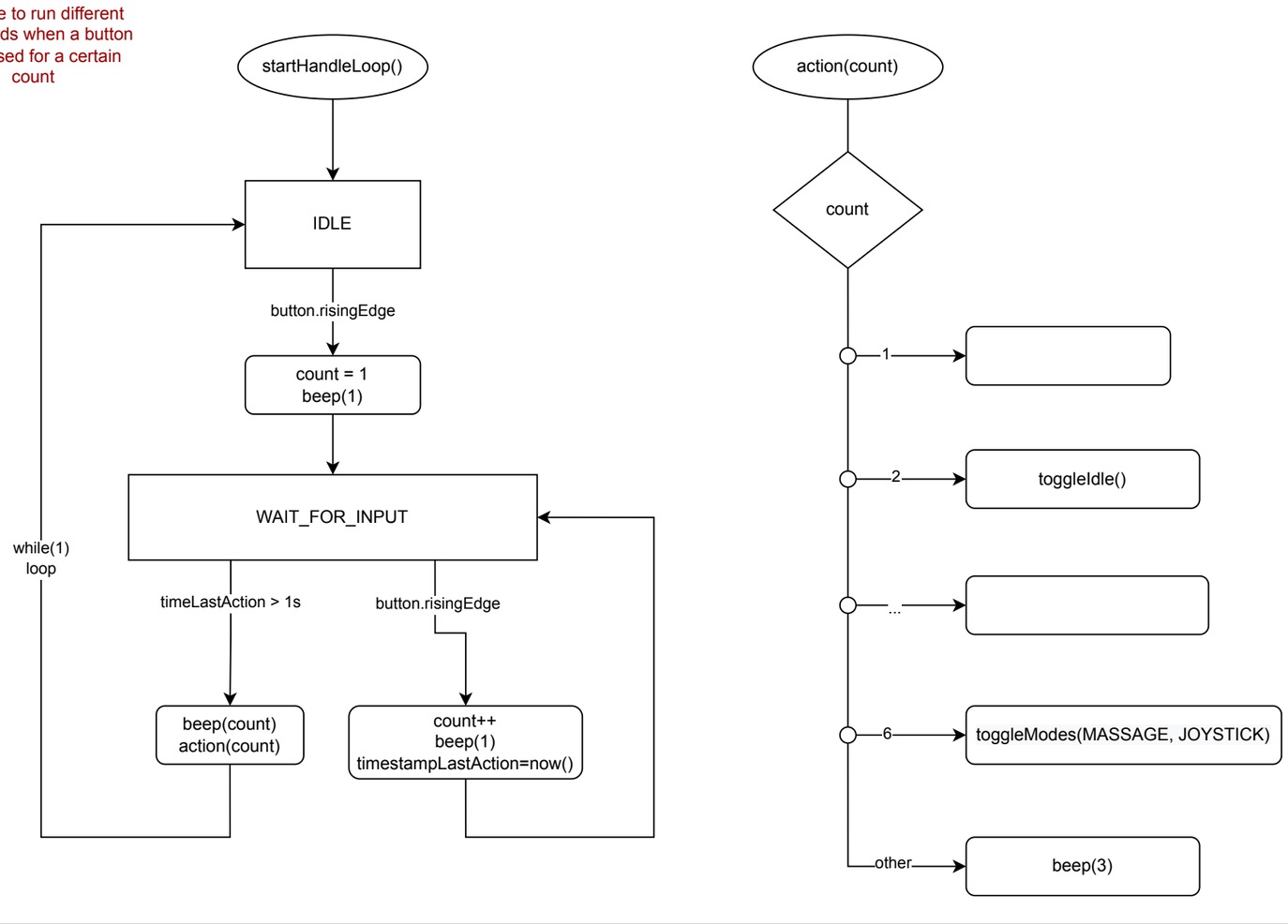
NOTE: the below diagrams might be significantly outdated -- needs update to match the actual code again

Tasks / Threads overview



class buttonCommands (button.hpp, button.cpp)

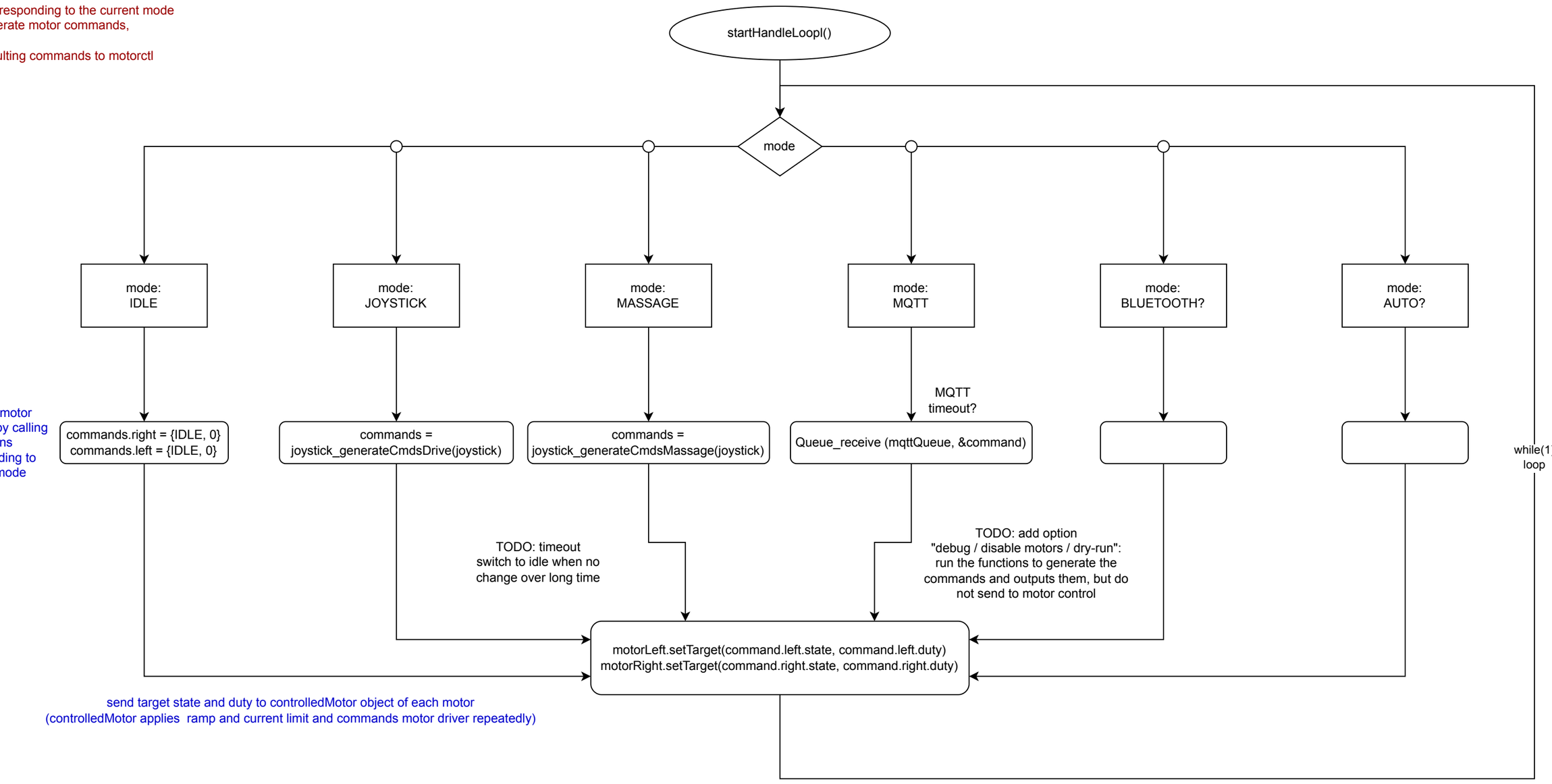
class which makes it possible to run different commands when a button is pressed for a certain count



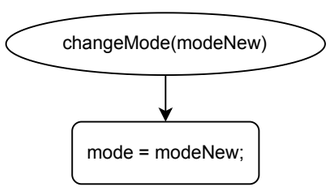
class controledArmchair (control.hpp, control.cpp)

run functions corresponding to the current mode that generate motor commands,

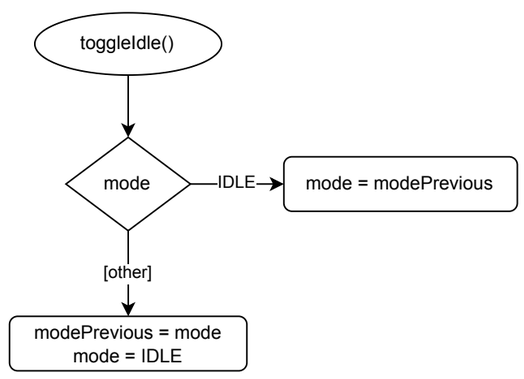
send the resulting commands to motorctl



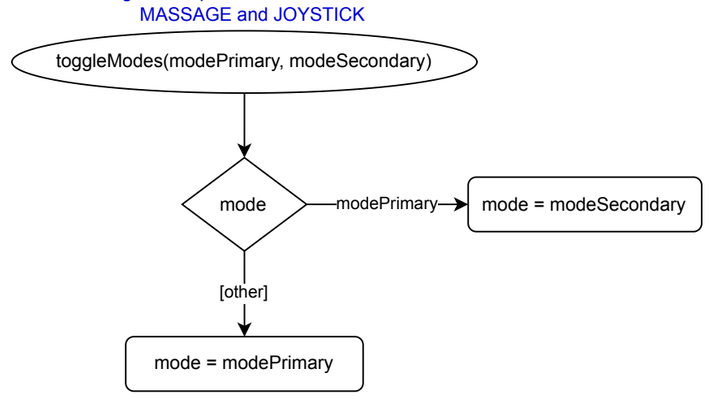
switch to specific mode



toggle between IDLE and previous or default mode



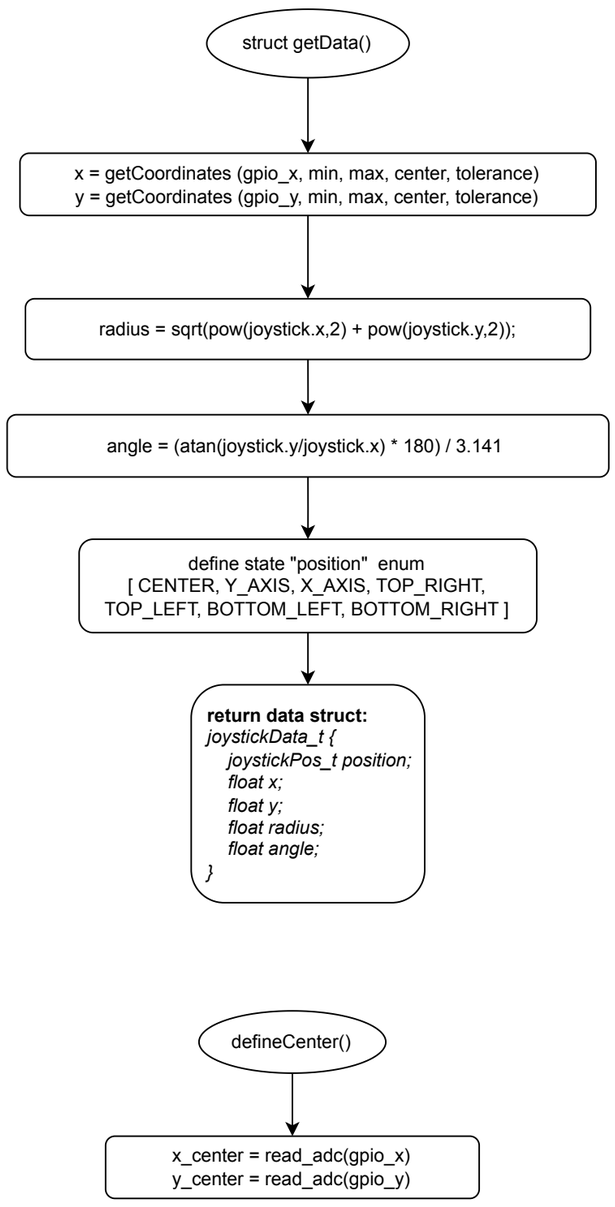
toggle between two specific modes
e.g. button press switches between MASSAGE and JOYSTICK



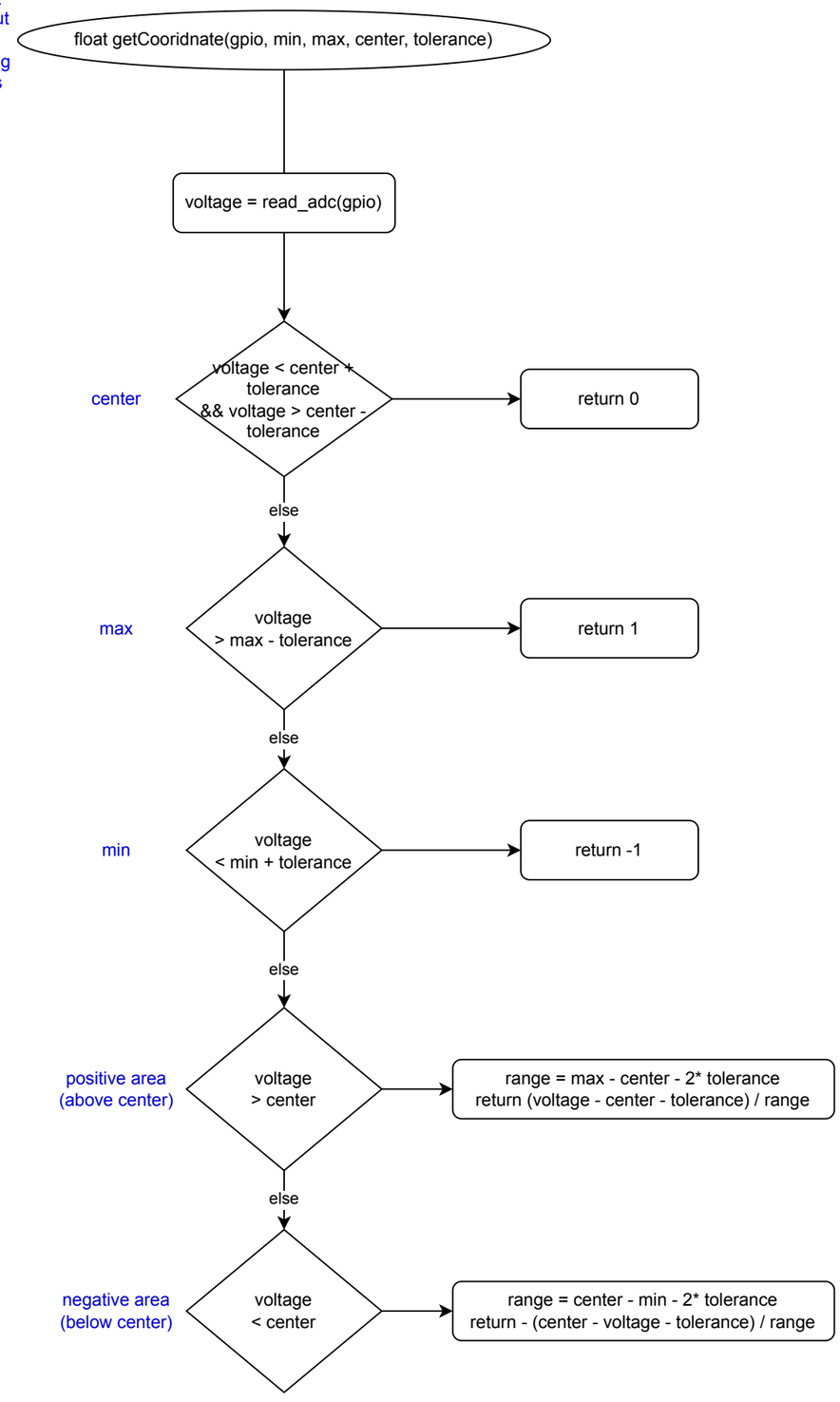
class evaluatedJoystick (joystick.hpp, joystick.cpp)

function that makes it possible to get a struct with current state and data of the joystick.

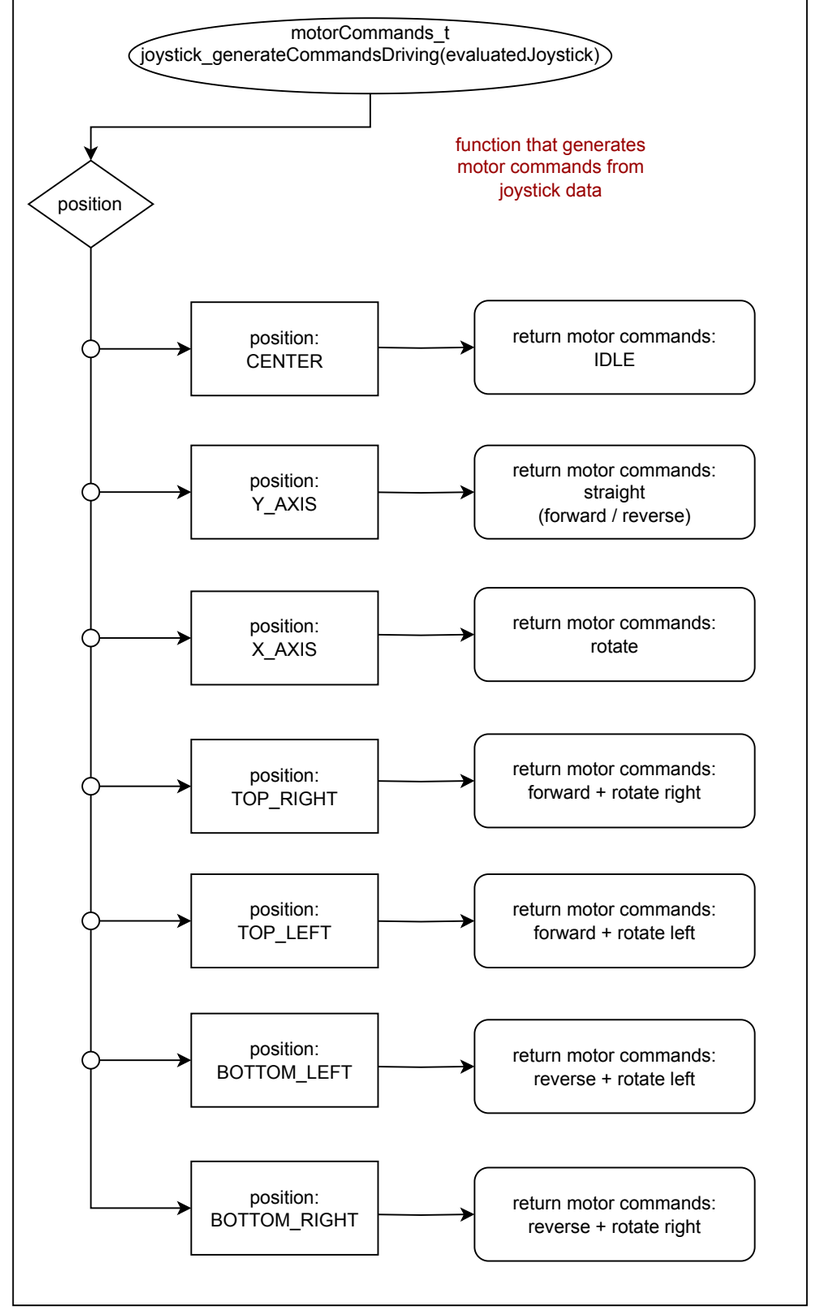
this can be used as input in other functions or tasks



private function that reads an analog input and calculates coodrinates according to given parameters

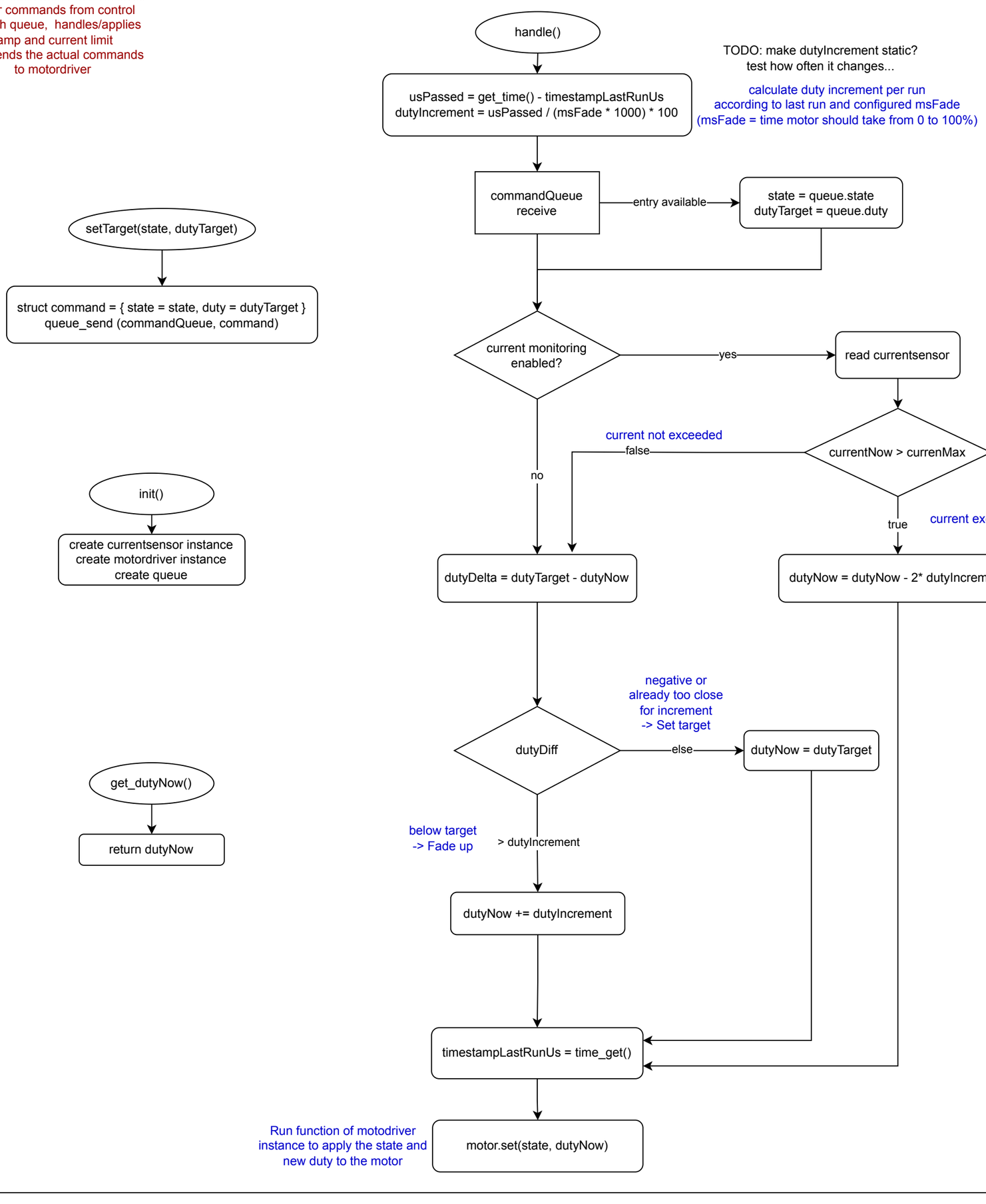


function: joystick_generateCommandsDriving

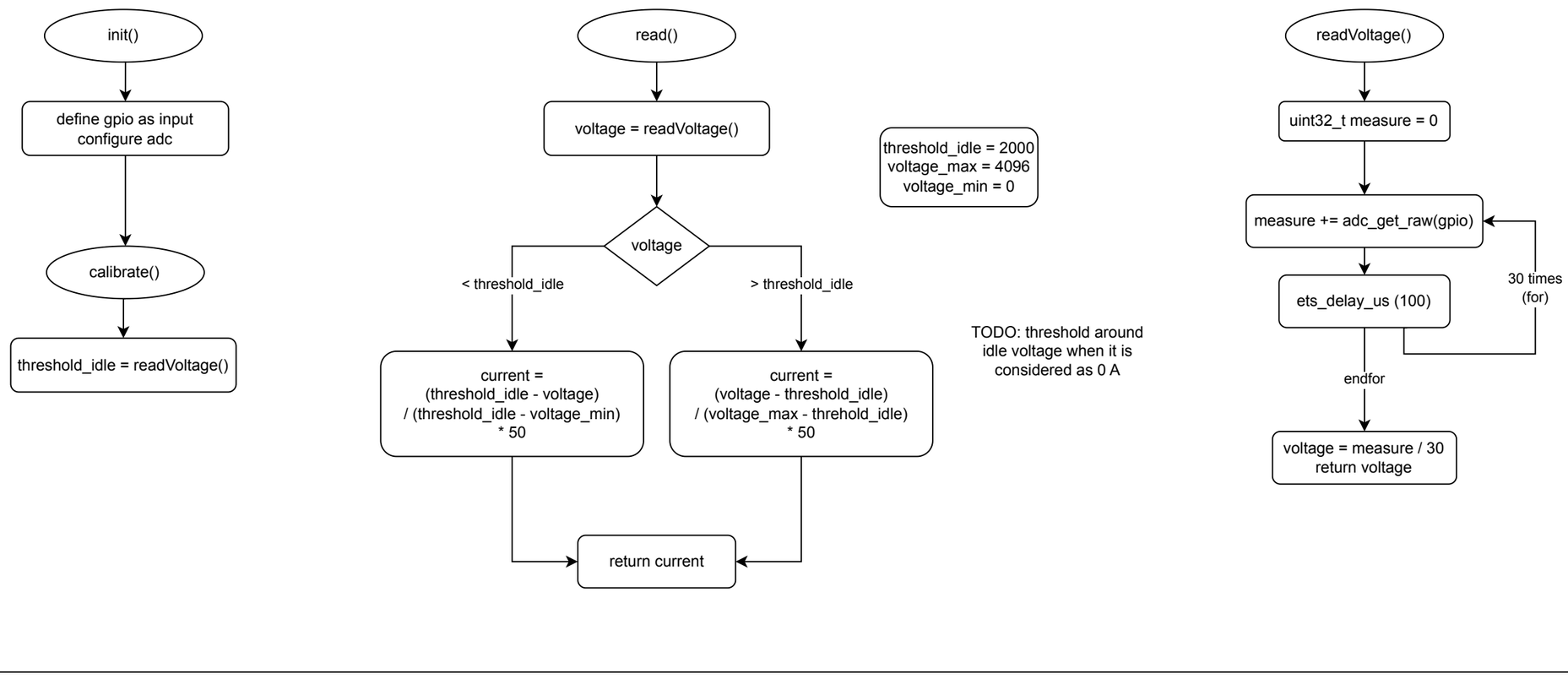


class controlledMotor (motorctl.hpp, motorctl.cpp)

class for each motor that receives motor commands from control through queue, handles/applies ramp and current limit then sends the actual commands to motordriver

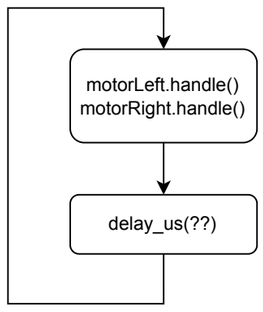


class currentsensor



DEL ### - old unneeded or temporary flowcharts

task motorCtl



global config

config.cpp only:
struct motorconfig_t config_motorLeft {... }
struct motorconfig_t config_motorRight {...}

config.hpp + config.cpp:
handledMotor **motorLeft** (& config_motorLeft);
handledMotor **motorRight** (& config_motorRight);