How to Write an Autonomous Function

An Autonomous function is a class that implements the AutFunction interface.

Using this interface will require three methods:

1. **public** **void** update(**long** deltaTime);
   1. Called repeatedly at deltaTime ms intervals until isDone() returns **true**.
2. **public void** init();
   1. Called once just before entering loop with update(**long** deltaTime). Useful when using sensors such encoders which may require an offset to be recorded before update(**long** deltaTime) is called.
3. **public** **boolean** isDone();
   1. Indicates that the Autonomous function has finished. update(**long** deltaTime) will stop being called and the next Autonomous function will begin.

The constructor for the Autonomous function is where the function will gain access to robot outputs such as the drive base, turret, intake, etc. This is done so that only certain mechanisms of the robot can be accessed by Autonomous functions. (Ex: Turn.java needs to access the drive base so the constructor includes a DriveBase object.

Let’s look at Turn.java as an example:

**public** **class** Turn **implements** AutFunction {

**private** **final** **double** angle;

**private** **final** ADXRS450\_Gyro gyro;

**private** **final** DriveBase base;

**private** **final** **double** p = 0.07;

**private** **final** **double** i = 0.00002;

**private** **final** **double** d = 0.02;

**private** **final** **double** acceptRange = 3;

**private** **final** **double** acceptRangeRate = 2;

**private** **double** turnIntegral;

**private** **boolean** isDone;

//Turns the robot to angle degrees

**public** Turn(DriveBase base, **double** angle) {

**this**.base = base;

isDone = **false**;

gyro = Robot.*sensors*.getGyro();

**if** (angle < 0) {

angle += 360;

}

// normalize angle

angle = angle - (**int**) (angle / 360) \* 360;

**this**.angle = angle;

}

**public** **void** update(**long** deltaTime) {

**double** curretAngle = gyro.getAngle();

**double** error = angle - curretAngle;

Robot.*nBroadcaster*.println(error);

// check if there is a faster way to get to the target by crossing the 0

// - 360 degrees jump thing

**if** (Math.*abs*(error - 360) < Math.*abs*(error)) {

error -= 360;

} **else** **if** (Math.*abs*(error + 360) < Math.*abs*(error)) {

error += 360;

}

turnIntegral += error;

**double** turnSpeed = error \* p - gyro.getRate() \* d + turnIntegral \* i;

base.move(-turnSpeed, turnSpeed);

**if** (Math.*abs*(error) < acceptRange

&& Math.*abs*(gyro.getRate()) < acceptRangeRate) {

isDone = **true**;

}

}

@Override

**public** **boolean** isDone() {

**return** isDone;

}

@Override

**public** **void** init() {

//nothing to initialize

}

**public** Turn(DriveBase base, **double** angle) { . . . }

The Constructor:

* Requires a DriveBase object so that the function can move the robot
* Requires a **double** which tells the robot what angle to turn to

**public** **void** update(**long** deltaTime) {

**double** curretAngle = gyro.getAngle();

**double** error = angle - curretAngle;

Robot.*nBroadcaster*.println(error);

// check if there is a faster way to get to the target by crossing the 0

// - 360 degrees jump thing

**if** (Math.*abs*(error - 360) < Math.*abs*(error)) {

error -= 360;

} **else** **if** (Math.*abs*(error + 360) < Math.*abs*(error)) {

error += 360;

}

turnIntegral += error;

**double** turnSpeed = error \* p - gyro.getRate() \* d + turnIntegral \* i;

base.move(-turnSpeed, turnSpeed);

**if** (Math.*abs*(error) < acceptRange

&& Math.*abs*(gyro.getRate()) < acceptRangeRate) {

isDone = **true**;

}

}

The update(**long** deltaTime) method:

* Contains the code that moves the robot
* Sets isDone to **true** once
  + The robot’s angle is within the acceptable range
  + The robot’s angular velocity is within the acceptable range

**public** **boolean** isDone() {

**return** isDone;

}

The isDone() method simply returns isDone which is changed in the update(**long** deltaTime) method.

**public** **void** init() {

//nothing to initialize

}

The init() method does nothing since this Autonomous function has nothing to initialize.

How to use the AutonomousManager

The AutonomousManager is responsible for running Autonomous Functions. It also ensures that the robot responds to commands such as Disabling the robot during autonomous. Putting a loop inside Autonomous functions is dangerous because an improperly written loop could cause the Robot to ignore Disable commands from the FMS. The structure of the AutonomousManager moves the process of checking if the robot is Disabled outside of individual functions, reducing the potential for catastrophic failure during autonomous.

Lines of code necessary to initialize and use AutonomousManager are as follow in order of execution:

AutonomousManager autManager = **new** AutonomousManager(clockRegulator);

The AutonomousManager constructor requires a ClockRegulator to define the rate that it runs an AutFunction at.

autManager.add(**new** Turn(base, 69));

To add an AutFunction to the autonomous routine, use the add(AutFunction f) method. Functions will be run in the order that they are added.

The above example adds a Turn function to the autonomous routine.

autManager.init();

Initializes AutonomousManager now that the autonomous routine is known

**while** (isAutonomous() && isEnabled() && !autManager.isDone()) {

autManager.update();

}

Runs the AutFunctions while the robot is in autonomous, is enabled, and hasn’t run all the AutFunctions.

**new** Stop(base, turret, intake).update(1);

Stops all mechanisms on the robot since Autonomous is over.

Here is an example of a complete implementation of AutonomousManager:

@Override

**public** **void** autonomous() {

*sensors*.getGyro().reset();

AutonomousManager autManager = **new** AutonomousManager(clockRegulator);

//Move the robot in a 36 inch square

autManager.add(**new** Turn(base, 0));

autManager.add(**new** MoveForwardInInches(base, 36));

autManager.add(**new** Turn(base, 270));

autManager.add(**new** MoveForwardInInches(base, 36));

autManager.add(**new** Turn(base, 180));

autManager.add(**new** MoveForwardInInches(base, 36));

autManager.add(**new** Turn(base, 90));

autManager.add(**new** MoveForwardInInches(base, 36));

autManager.add(**new** Turn(base, 0));

autManager.init();

**while** (isAutonomous() && isEnabled() && !autManager.isDone()) {

autManager.update();

}

**new** Stop(base, turret, intake).update(1);

Robot.*nBroadcaster*.println("End of autonomous");

}