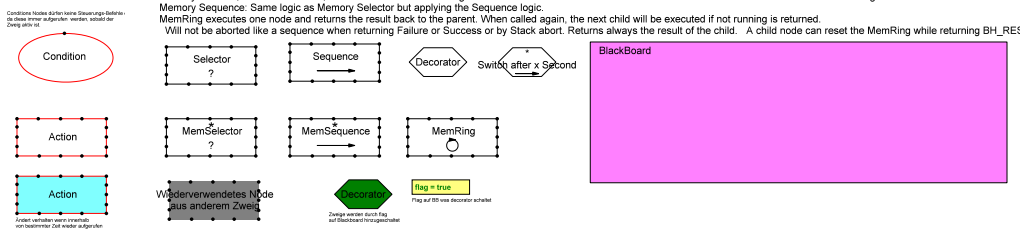
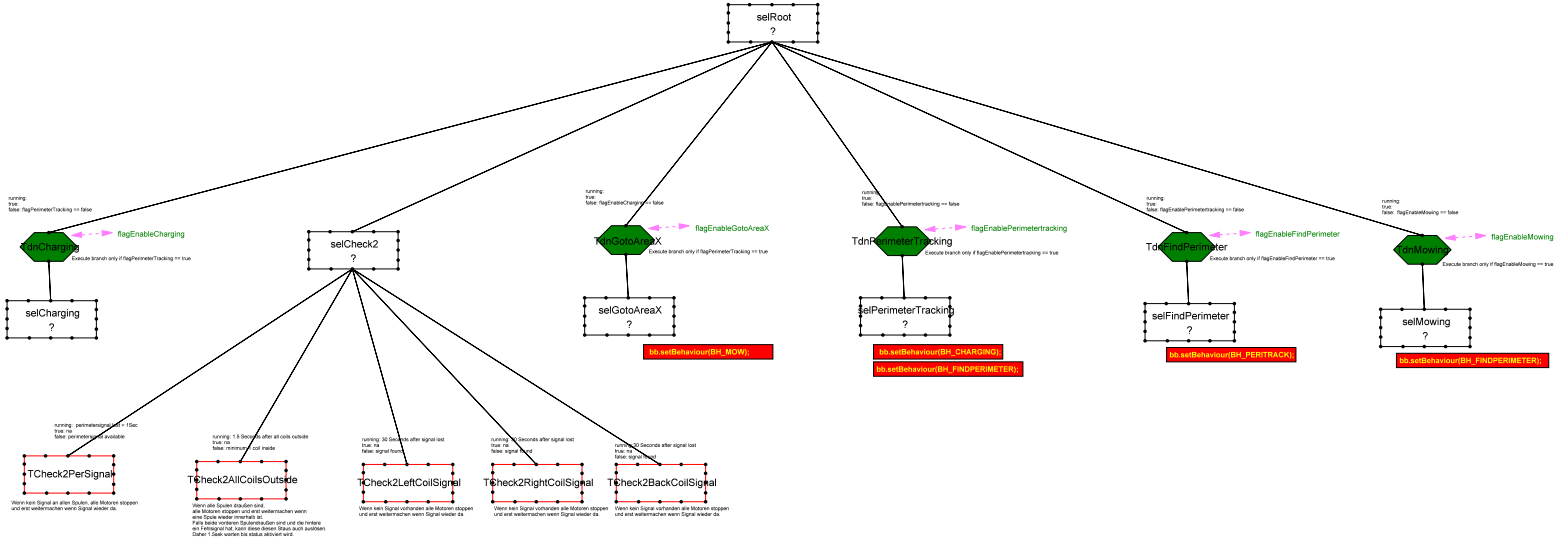
Selector: Execute every child till one returns Success or Running. If none then it return Failure. In other words, it returns the value of the first child that complete his logic.
Sequence: Execute every child unless one of them return failure.
Memory Selector; Same as Selector, but if one child return Running then at the next AI turn the BT will resume its execution on the node that returned Running.
Memory Sequence: Same logic as Memory Selector but applying the Sequence logic.
MemRing executes one node and returns the result back to the parent. When called again, the next child will be executed if not running is returned.
Will not be aborted like a sequence when returning Failure or Success or by Stack abort. Returns always the result of the child. A child node can reset the MemRing while returning BH_RESETx





bb.setBehaviour(BH_PERITRACK);

Schaltet die decorators hinter selRoot und ändert damit das Verhalten. Ein Zweig aktiviert somit einen anderen Zweig oder eine Änderung findet über das UserInterface statt.

```
void setBehaviour(enuBehaviour b ) {

  flagEnableMowing = false;
  flagEnablePerimetertracking = false;
  flagEnableCharging = false;
  flagEnableGotoAreaX = false;
  flagEnableFindPerimeter = false;
  motor.enableDefaultRamping();


  switch(b) {
     case BH_GOTOAREA:
        flagEnableGotoAreaX = true;
        rangeSensor.enabled = false;
        chargeSystem.deactivateRelay();  //Muss hier au
        motor.mowMotStop(); // Will be started in BH_M
        debug->printf("Set BH_GOTOAREA\r\n");
        break;
     case BH_CHARGING:
        flagEnableCharging = true;
        rangeSensor.enabled = false;
        motor.mowMotStop();
        debug->printf("Set BH_CHARGING\r\n");
        break;
     case BH_PERITRACK:
        flagEnablePerimetertracking = true;
        rangeSensor.enabled = false;
        chargeSystem.deactivateRelay();
        motor.mowMotStop();
        debug->printf("Set BH_PERITRACK\r\n");
        break;
     case BH_FINDPERIMETER:
        flagEnableFindPerimeter = true;
        rangeSensor.enabled = true;
        chargeSystem.deactivateRelay();
        motor.mowMotStop();
        debug->printf("Set BH_FINDPERIMETER\r\n");
        break;
     case BH_MOW:
        flagEnableMowing = true;
        rangeSensor.enabled = true;
        chargeSystem.deactivateRelay();
        debug->printf("Set BH_MOW\r\n");
        break;
     case BH_NONE:
        rangeSensor.enabled = false;
        chargeSystem.deactivateRelay();
        motor.mowMotStop();
        debug->printf("Set BH_NONE\r\n");
        break;
     default:
        debug->printf("setBehaviour unbekanntes Beha
  }
}
```

Zu beachten: wenn die aktuelle Node etwas setzt z.B: enablePerTrackRamping();
und der tree dann nach root zurückkehrt und vor Aufruf der aktuellen Node
war eine andere Node im running status, so wird diese nun vom Stack beendet und onTerminate aufgerufen
NACHDEM die aktuelle Node ausgeführt wurde. Wenn nun in onTerminate z.B. enableDefaultRamping()
aufgerufen wird, so wird enablePerTrackRamping() der aktuellen Node wieder rückgängig gemacht.

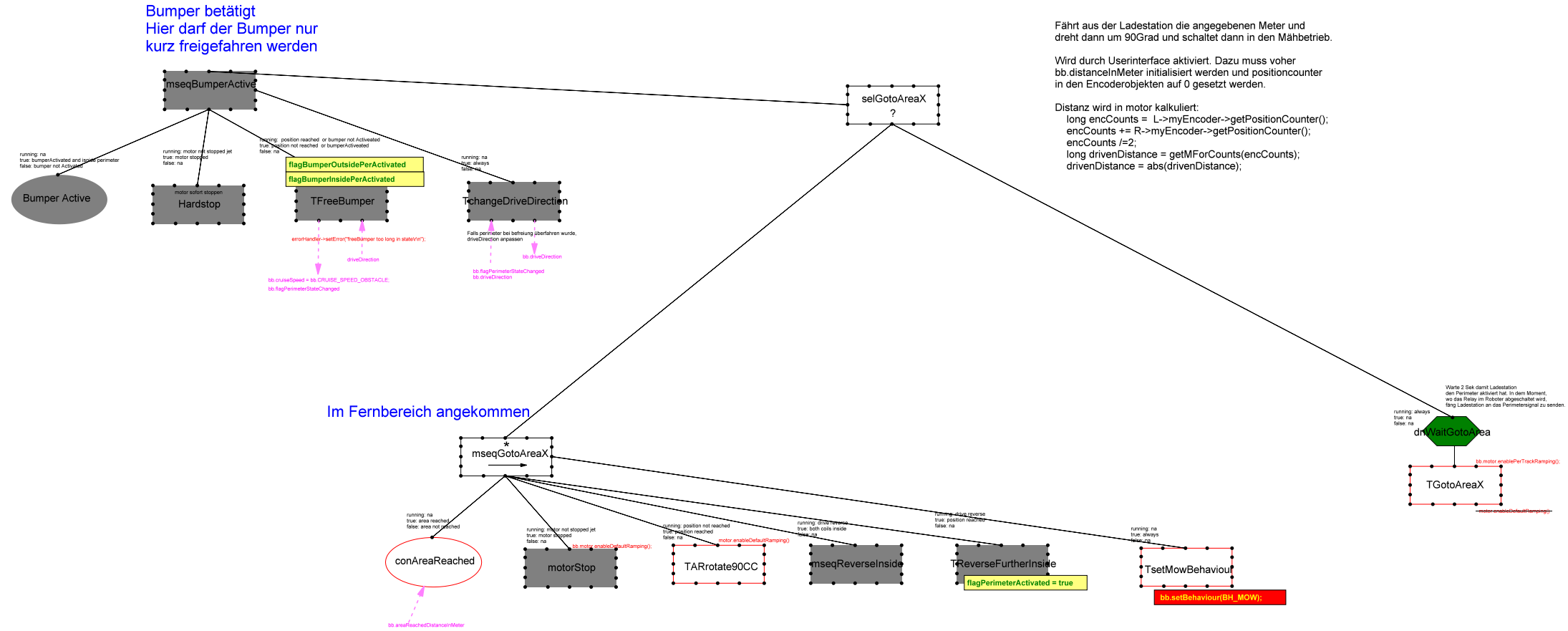
Daher wurde motor.enableDefaultRamping() aus onTerminate rausgenommen.
Wenn das behaviour mit setBehaviour(enuBehaviour b) verändert wird, wird motor.enableDefaultRamping();
als default gesetzt. Der Tree ist dann in jedem Zweig dafür verantwortlich, wie er das ramping umschaltet.

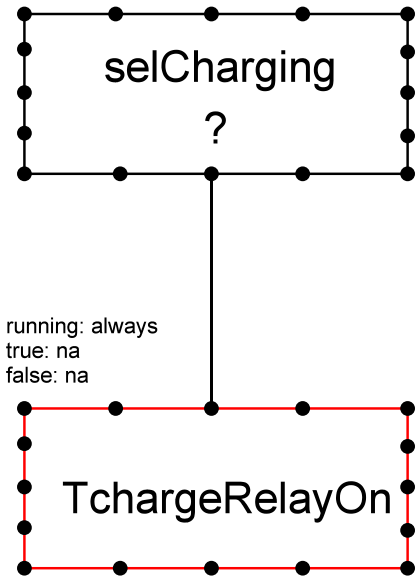
Man könnte in onTeminate if(status != BH_ABORTED) abfragen, dann wird der Befehl nur ausgeführt, wenn
sich die Node selber beendet. Bring hier aber nicht, da bei Verlassen der Node auf enableDefaultRamping()
zurückgesetzt werden soll.

Eigentlich müsste jede Node, die die Geschwindigkeit setzt, das für sich benötigte Ramping einstellen.

Was ist, wenn eine node in einem neuen Zweig ist die wiederverwendet wird aber auch auf dem alten Stack
im Status running ist?
Bei Aufruf der Node wird onInitialize nicht aufgerufen, da die node bereits in running ist.
D.h. onUpdate wird ausgeführt und bei der Rückkehr zum root befindet sich die node immer noch
im NodeStack lastRunningNodes; als running und wird n un zurückgesetzt!!! Beim nächsten Aufruf wird dann
onInitialize aufgerufen und die Node geht wieder in running.

reusability: due to the independence of nodes in BT, the subtrees are also independent.
This allows the reuse of nodes or subtrees among other trees or projects.
Das gilt nur, solange keine globalen werte verändert werden in onTerminate.
Falls doch muss berücksichtig werden, ob onTerminate bei Aufruf nicht gerade was zurücksetzt,
was eine andere Node gesetzt hat.

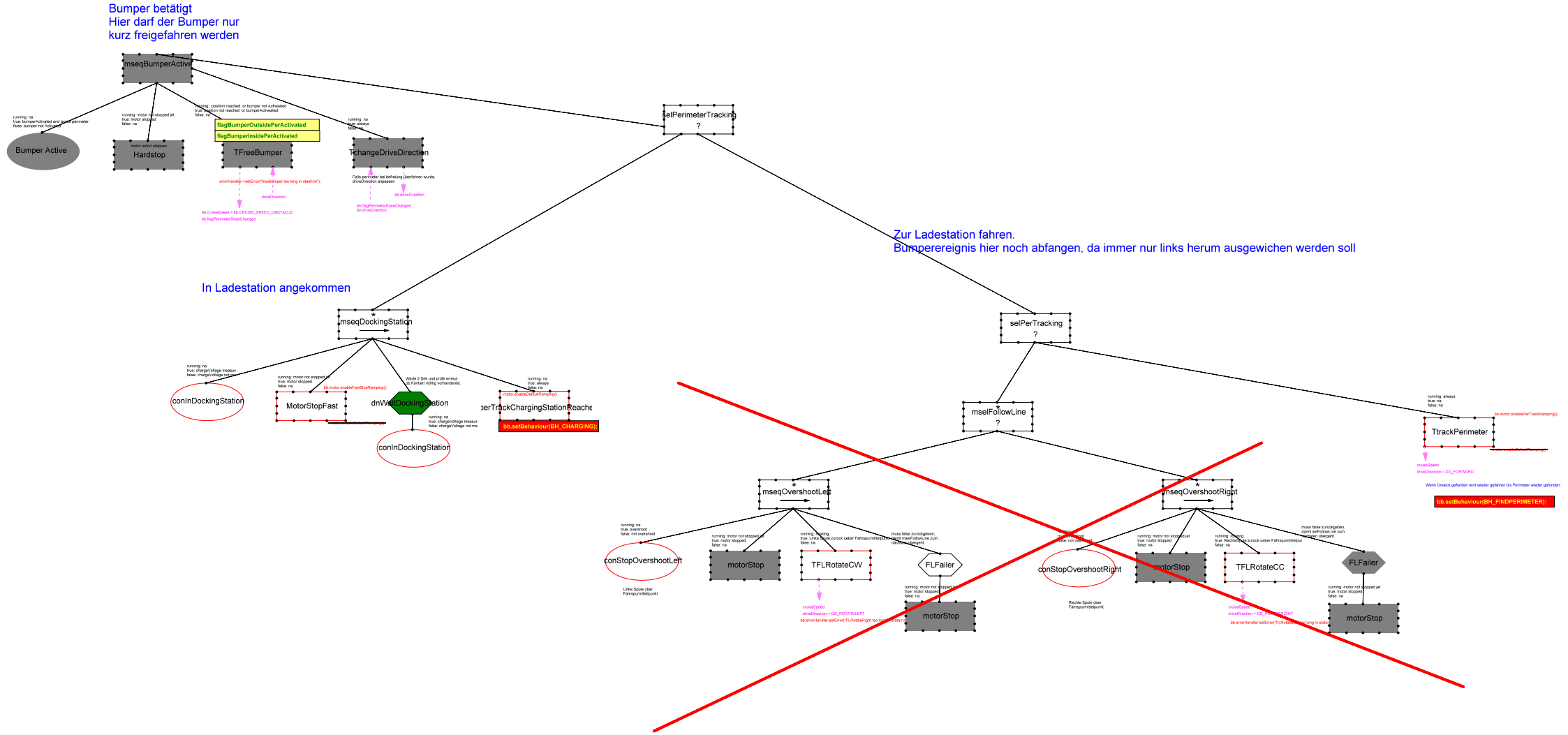

Bumper betätigt
Hier darf der Bumper nur
kurz freigefahren werden
mseqBumperActive
running: na
true: bumperActivated and isInsidePerimeter
false: bumper not Activated
Bumper Active
running: motor not stopped jet
true: motor stopped
false: na
motor sofort stoppen
Hardstop
running: position reached or bumper not Activeated
true: position not reached or bumperActiveated
false: na
flagBumperOutsidePerActivated
flagBumperInsidePerActivated
TFreeBumper
errorHandler->setError("freeBamper too long in state\n\n");
driveDirection
bb.cruiseSpeed = bb.CRUISE_SPEED_OBSTACLE;
bb.flagPerimeterStateChanged
running: na
true: always
false: na
TchangeDriveDirection
Falls perimeter bei befreiung überfahren wurde,
driveDirection anpassen
bb.driveDirection
bb.flagPerimeterStateChanged
bb.driveDirection
selGotoAreaX
?
Fährt aus der Ladestation die angegebenen Meter und
dreht dann um 90Grad und schaltet dann in den Mähbetrieb.
Wird durch Userinterface aktiviert. Dazu muss voher
bb.distanceInMeter initialisiert werden und positioncounter
in den Encoderobjekten auf 0 gesetzt werden.
Distanz wird in motor kalkuliert:
long encCounts = L->myEncoder->getPositionCounter();
encCounts += R->myEncoder->getPositionCounter();
encCounts /=2;
long drivenDistance = getMForCounts(encCounts);
drivenDistance = abs(drivenDistance);
Im Fernbereich angekommen
*
mseqGotoAreaX
running: na
true: area reached
false: area not reached
conAreaReached
bb.areaReachedDistanceInMeter
running: motor not stopped jet
true: motor stopped
false: na
bb.motor.enableDefaultRamping();
motorStop
running: position not reached
true: position reached
false: na
motor.enableDefaultRamping()
TARrotate90CC
running: drive reverse
true: both coils inside
false: na
mseqReverseInside
running: drive reverse
true: position reached
false: na
TReverseFurtherInside
flagPerimeterActivated = true
running: na
true: always
false: na
TsetMowBehaviour
bb.setBehaviour(BH_MOW);
Warte 2 Sek damit Ladestation
den Perimeter aktiviert hat. In dem Moment,
wo das Relay im Roboter abgeschaltet wird,
fängt Ladestation an das Perimetersignal zu senden.
running: always
true: na
false: na
dmWaitGotoArea
bb.motor.enablePerTrackRamping();
TGotoAreaX
motor.enableDefaultRamping();

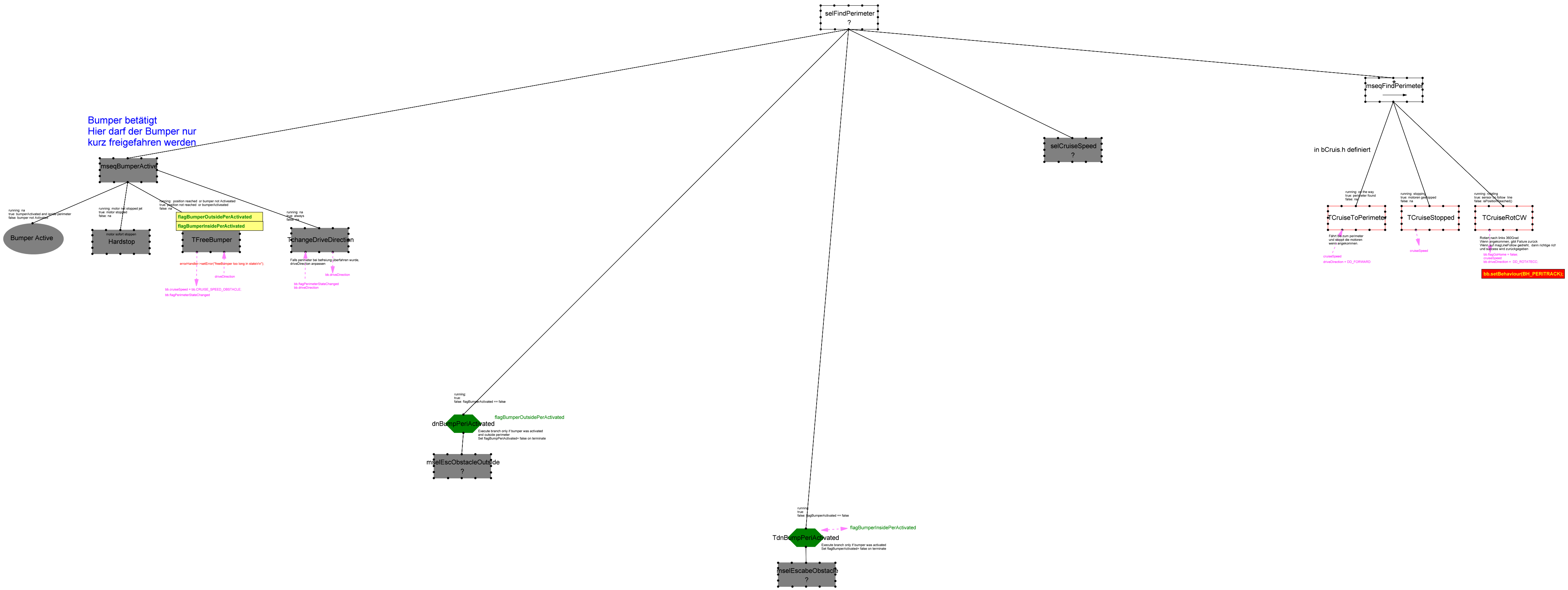


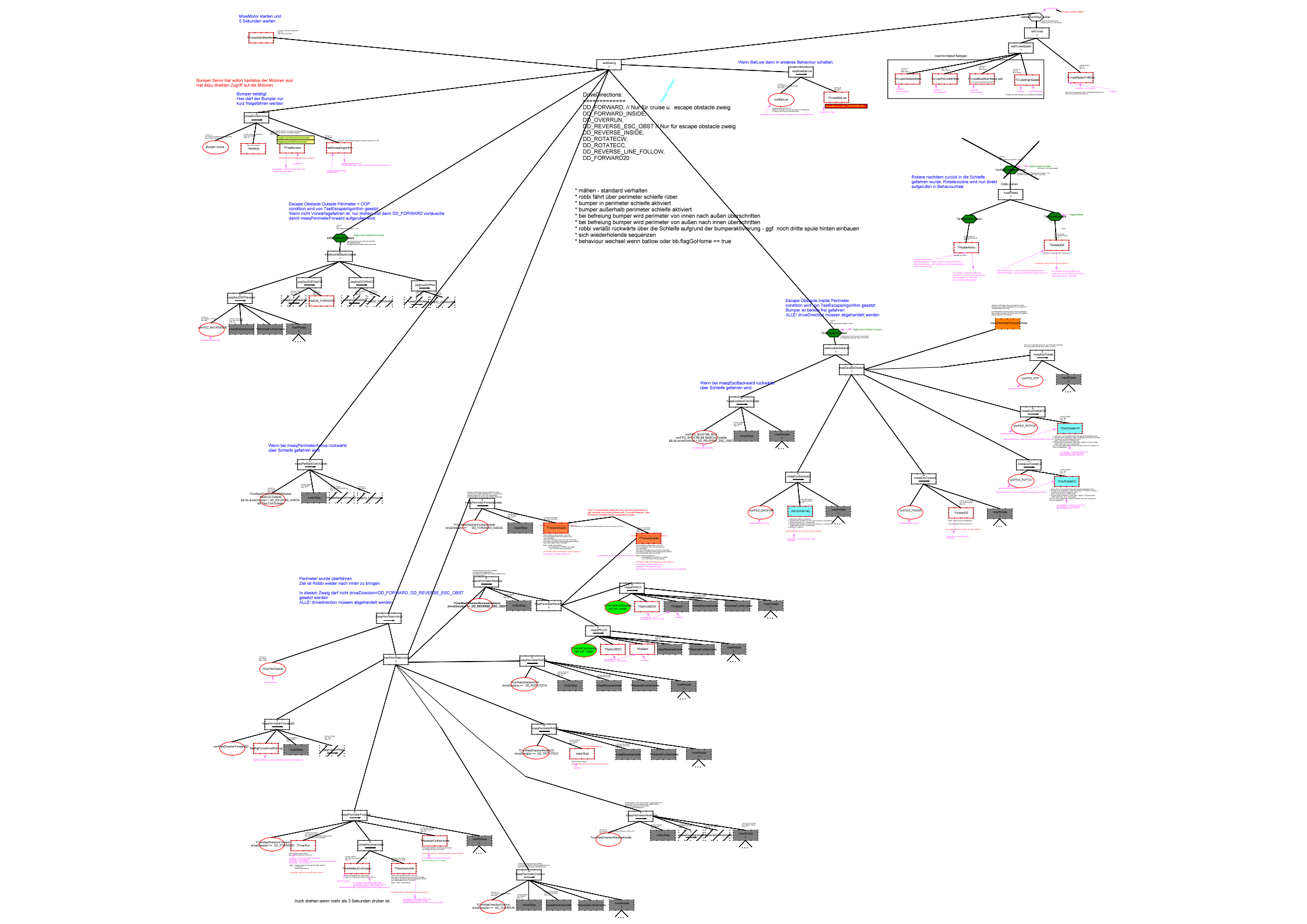
Schaltet Relay an.
Schaltet Relay ab wenn onTerminate aufgerufen wird.

Relay wird auch in setBehaviour( )
abgeschaltet, bei Wechsel in ein Behaviour, dass nicht BH_CHARGING ist.
Grund: falls user dieses über das Userinterface eingeschaltet hat.
Relay zieht nur in der Ladestation an!

//Muss hier auch abgeschaltet werden, falls user im Manuel mode dieses einschaltet.


Bumper betätigt
Hier darf der Bumper nur
kurz freigefahren werden
mseqBumperActivat
running: na
true: bumperActivated and inside perimeter
false: bumper not Activated
Bumper Active
running: motor not stopped jet
true: motor stopped
false: na
motor sofort stoppen
Hardstop
running: position reached or bumper not Activated
true: position not reached or bumperActivated
false: na
flagBumperOutsidePerActivated
flagBumperInsidePerActivated
TFreeBumper
errorHandler->setError("freeBumper too long in stateVar");
driveDirection
bb.cruiseSpeed = bb.CRUISE_SPEED_OBSTACLE;
bb.flagPerimeterStateChanged
running: na
true: always
false: na
TchangeDriveDirection
Falls perimeter bei befreiung überfahren wurde,
driveDirection anpassen
bb.driveDirection
bb.flagPerimeterStateChanged
bb.driveDirection
selPerimeterTracking
?
In Ladestation angekommen
mseqDockingStation
running: na
true: chargerVoltage measur
false: chargerVoltage not me
conInDockingStation
running: motor not stopped jet
true: motor stopped
false: na
bb.motor.enableFastStopRamping();
MotorStopFast
Warte 2 Sek und prüfe erneut
ob Kontakt richtig vorhanden ist.
dnWaitDockingStation
running: na
true: chargerVoltage measur
false: chargerVoltage not me
conInDockingStation
running: na
true: always
false: na
motor.disableFastRamping();
erTrackChargingStationReache
bb.setBehaviour(BH_CHARGING);
Zur Ladestation fahren.
Bumperereignis hier noch abfangen, da immer nur links herum ausgewichen werden soll
selPerTracking
?
mselFollowLine
?
mseqOvershootLeft
running: na
true: overshoot
false: not overshoot
conStopOvershootLeft
Linke Spule über
Fahrspurmittelpunkt
running: motor not stopped jet
true: motor stopped
false: na
motorStop
running: rotating
true: Linke Spule zurück ueber Fahrspurmittelpunkt
false: na
TFLRotateCW
cruiseSpeed
driveDirection = DD_ROTATELEFT
bb.errorHandler.setError("FLRotateRight too long in stateVar");
muss false zurückgeben,
damit mselFollowLine zum
nächsten Zustand wechselt
FLFailer
running: motor not stopped jet
true: motor stopped
false: na
motorStop
mseqOvershootRight
running: na
true: overshoot
false: not overshoot
conStopOvershootRight
Rechte Spule über
Fahrspurmittelpunkt
running: motor not stopped jet
true: motor stopped
false: na
motorStop
running: rotating
true: Rechte Spule zurück ueber Fahrspurmittelpunkt
false: na
TFLRotateCC
cruiseSpeed
driveDirection = DD_ROTATERIGHT
bb.errorHandler.setError("FLRotateRight too long in stateVar");
muss false zurückgeben,
damit selFollowLine zum
nächsten Übergang
FLFailer
running: motor not stopped jet
true: motor stopped
false: na
motorStop
running: always
true: na
false: na
TtrackPerimeter
bb.motor.enablePerTrackRamping();
cruiseSpeed
driveDirection = DD_FORWARD
Wenn Dreieck gefunden wird wieder gefahren bis Perimeter wieder gefunden:
bb.setBehaviour(BH_FINDPERIMETER);


selFindPerimeter
?
Bumper betätigt
Hier darf der Bumper nur
kurz freigefahren werden
mseqBumperActive
running: na
true: bumperActivated and isnide perimeter
false: bumper not Activated
Bumper Active
running: motor not stopped jet
true: motor stopped
false: na
motor sofort stoppen
Hardstop
running: position reached or bumper not Activeated
true: position not reached or bumperActiveated
false: na
flagBumperOutsidePerActivated
flagBumperInsidePerActivated
TFreeBumper
errorHandler->setError("freeBumper too long in state\r\n");
driveDirection
bb.cruiseSpeed = bb.CRUISE_SPEED_OBSTACLE;
bb.flagPerimeterStateChanged
running: na
true: always
false: na
TchangeDriveDirection
Falls perimeter bei befreiung überfahren wurde,
driveDirection anpassen
bb.driveDirection
bb.flagPerimeterStateChanged
bb.driveDirection
running:
true:
false: flagBumperActivated == false
dnBumpPeriActivated
flagBumperOutsidePerActivated
Execute branch only if bumper was activated
and outside perimeter
Set flagBumpPeriActivated= false on terminate
mselEscObstacleOutside
?
running:
true:
false: flagBumperActivated == false
TdnBumpPeriActivated
flagBumperInsidePerActivated
Execute branch only if bumper was activated
Set flagBumperActivated= false on terminate
mselEscabeObstacle
?
selCruiseSpeed
?
mseqFindPerimeter
in bCruis.h definiert
running: on the way
true: perimeter found
false: na
TCruiseToPerimeter
Fährt bis zum perimeter
und stoppt die motoren
wenn angekommen.
cruiseSpeed
driveDirection = DD_FORWARD
running: stopping
true: motoren gestopped
false: na
TCruiseStopped
cruiseSpeed
running: rotating
true: sensor on follow line
false: isPositionReached()
TCruiseRotCW
Rotiert nach links 360Grad
Wenn angekommen, gibt Failure zurück
Wenn auf magLineFollow gedreht, dann richtige rich
und success wird zurückgegeben
bb.flagGoHome = false;
cruiseSpeed
bb.driveDirection = DD_ROTATECC;
bb.setBehaviour(BH_PERITRACK);


MowMotor starten und
5 Sekunden warten.
Bumper Sensor löst sofort handstop der Motoren aus!
Hat dazu direkten Zugriff auf die Motoren.
Bumper betätigt
Hier darf der Bumper nur
kurz freigefahren werden
Wenn BatLow dann in anderes Behaviour schalten.
DriveDirections:
==========
DD_FORWARD, // Nur für cruise u. escape obstacle zweig
DD_FORWARD_INSIDE,
DD_OVERRUN,
DD_REVERSE_ESC_OBST // Nur für escape obstacle zweig
DD_REVERSE_INSIDE,
DD_ROTATECW,
DD_ROTATECC,
DD_REVERSE_LINE_FOLLOW,
DD_FORWARD20
* mähen - standard verhalten
* robbi fährt über perimeter schleife rüber
* bumper in perimeter schleife aktiviert
* bumper außerhalb perimeter schleife aktiviert
* bei befreiung bumper wird perimeter von innen nach außen überschritten
* bei befreiung bumper wird perimeter von außen nach innen überschritten
* robbi verläßt rückwärts über die Schleife aufgrund der bumperaktivierung - ggf. noch dritte spule hinten einbauen
* sich wiederholende sequenzen
* behaviour wechsel wenn batlow oder bb.flagGoHome == true
Escape Obstacle Outside Perimeter = OOP
condition wird von TaskEscapeAlgorithm gesetzt
Wenn nicht Vorwärtsgefahren ist, nur drehen und dann DD_FORWARD vortäusche
damit mseqPerimeterForward aufgerufen wird.
Wenn bei mseqPerimeterActive rückwärts
über Schleife gefahren wird
Escape Obstacle Inside Perimeter
condition wird von TaskEscapeAlgorithm gesetzt
Bumper ist bereits frei gefahren
ALLE! driveDirection müssen abgehandelt werden
Wenn bei mseqEscBackward rückwärts
über Schleife gefahren wird
Rotiere nachdem zurück in die Schleife
gefahren wurde. Rotateroutine wird nun direkt
aufgerufen in Behaviourtree
Perimeter wurde überfahren
Ziel ist Robbi wieder nach innen zu bringen
In diesem Zweig darf nicht driveDirection=DD_FORWARD, DD_REVERSE_ESC_OBST
gesetzt werden
ALLE! drivedirection müssen abgehandelt werden
Auch drehen wenn mehr als 3 Sekunden drüber ist.