Moonlight User Manual

2023-03-19

Inhalt

- Einführung
- Moonlight Command Overview
- Ardumower Control Program – **amcp**
- Ardumower Command Line Tool – **amcmd**
- Maps

Moonlight Firmware

- Moonlight ist der Name meiner modifizierten Sunray firmware (fork von Sunray-Version V168)

- Moonlight Features:
  - verbessertes Ausgabeformat für Logging
  - kann Kommandos über UDP verarbeiten
  - unterstützt Logging über UDP
  - enthält einen Simulationsmodus der mit einem Arduino Due + SD Card + ESP8266 + RTC arbeitet (für Software-Tests ohne Ardumower)
  - unterstützt zusätzliche Kommandos
  - unterstützt RTC (Logging Timestamp & SD Card File Date)
  - Unterstützt SD-Card Filesystem Kommandos (ls, rm, cat, cp)
  - Unterstützt Remote Control via UDP (#-Kommando)

Moonlight Command Overview
(Sunray Commands)

GUI	Moonlight Command[1]	Short Description
	AT+M, *fLinear, fAngular*	Set Motor Linear & Angular Speed
i,m, d,a[4]	AT+C, *bEnableMowMotor, iOperationType[2], fSpeed, iFixTimeout, bFinishAndRestart, fMowingPointPercent, bSkipNextMowingPoint, bEnableSonar*	Mower Control
	AT+W, *iIndex, fX, fY*	Set Waypoint
	AT+N, *iWayPerimeter, iWayExclusion, iWayDock, iWayMow, iWayFree*	Set Waypoints Count
	AT+X, *0[, iExclusionLength0[, 1, iExclusionLength1[, …]]]*	ExclusionCounts
	AT+P, *bAbsolutePosSource, fLongitude, fLatitude*	Set Position
	AT+S	Print Summary
v	AT+V	Print Version Number
s	AT+T	Print Statistics
	AT+L	Clear Statistics
	AT+E	Motor Test
	AT+O	Trigger Obstacle
	AT+F	Sensor Test
	AT+G	Toggle GPS.Solution[3]
	AT+K	Kidnapping Test
	AT+Z	Stress Test
	AT+Y	Trigger Watchdog Test
	AT+Y2	GNSS Reboot

Notes:

(1) Parameter naming convention: *fName*=float, *iName*=int, *bName*=bool (0 or 1)

(2) *OperationType* = OP_IDLE, OP_MOW, OP_CHARGE, OP_ERROR, OP_DOCK

(3) GPS.Solution = SOL_INVALID=0, SOL_FLOAT=1, SOL_FIXED=2

(4) i=Idle, m=start mowing, d=start docking, a=start undocking

Moonlight Command Overview
(Moonlight Extensions)

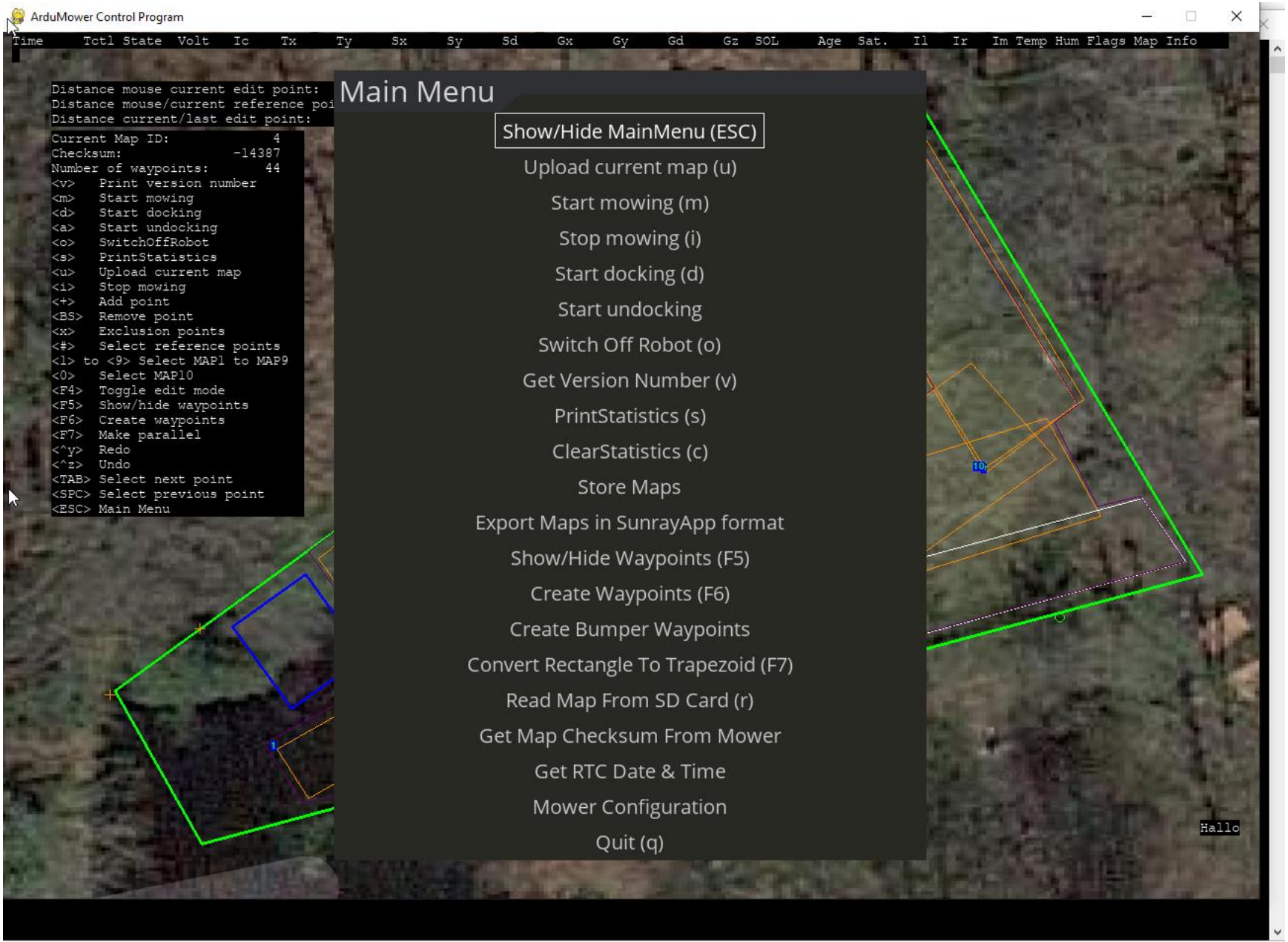
GUI	Moonlight Command[(1)]	Short Description
	AT+A,*iMinElev, iNumSV, iMinCN0*	Upload GPS Config Filter
	AT+U,*CurrentMapIndex*	Start Upload Map (informs Moonlight about *CurrentMapIndex* and initializes map file)
	AT+R,*MapIndex*	Read map from SD Card. The file name is „MAP<*MapIndex*>.TXT“. *MapIndex* = 1…10
	AT+D[,[yy]*yy,mo,dd,hh,mi,ss,w*]	Get/set RTC date and time (w=weekday 0-6; 0=Sunday)
o	AT+Y3	Switch Off Robot
	AT+Y4	Trigger Raspi Shutdown
b	AT+Y5	Toggle Bluetooth Logging
	AT+Y6	Ping (no operation, just reset auto power off counter)
	AT+YR	GNSS Hardware Reset
	AT+YS	Toggle SMOOTH_CURVES
p	AT+YP	Toggle UseGPSfloatForPosEstimation
	AT+YD	Toggle UseGPSfloatForDeltaEstimation
	AT+YG	Toggle GPS Logging
	AT+$L[,*fileName*]	Filesystem: ls *fileName* (‘*‘ can be used at end of *filename* as wildcard)
	AT+$C,*fileName*	Filesystem: cat *fileName* (‘*‘ can be used at end of *filename* as wildcard)
	AT+$T	Filesystem: ls –R (list all files in folder tree; output to Serial only)
	AT+$R,*fileName*	Filesystem: rm *fileName* (‘*‘ can be used at end of *filename* as wildcard)
	AT+#,*iLinear, iAngular*	Remote control: Set linear and angular speed (range -2048 to 2048) – no response

Notes:
(1) Parameter naming convention: *fName*=float, *iName*=int, *bName*=bool (0 or 1)
(2) *OperationType* = OP_IDLE, OP_MOW, OP_CHARGE, OP_ERROR, OP_DOCK
(3) GPS.Solution = SOL_INVALID, SOL_FLOAT, SOL_FIXED

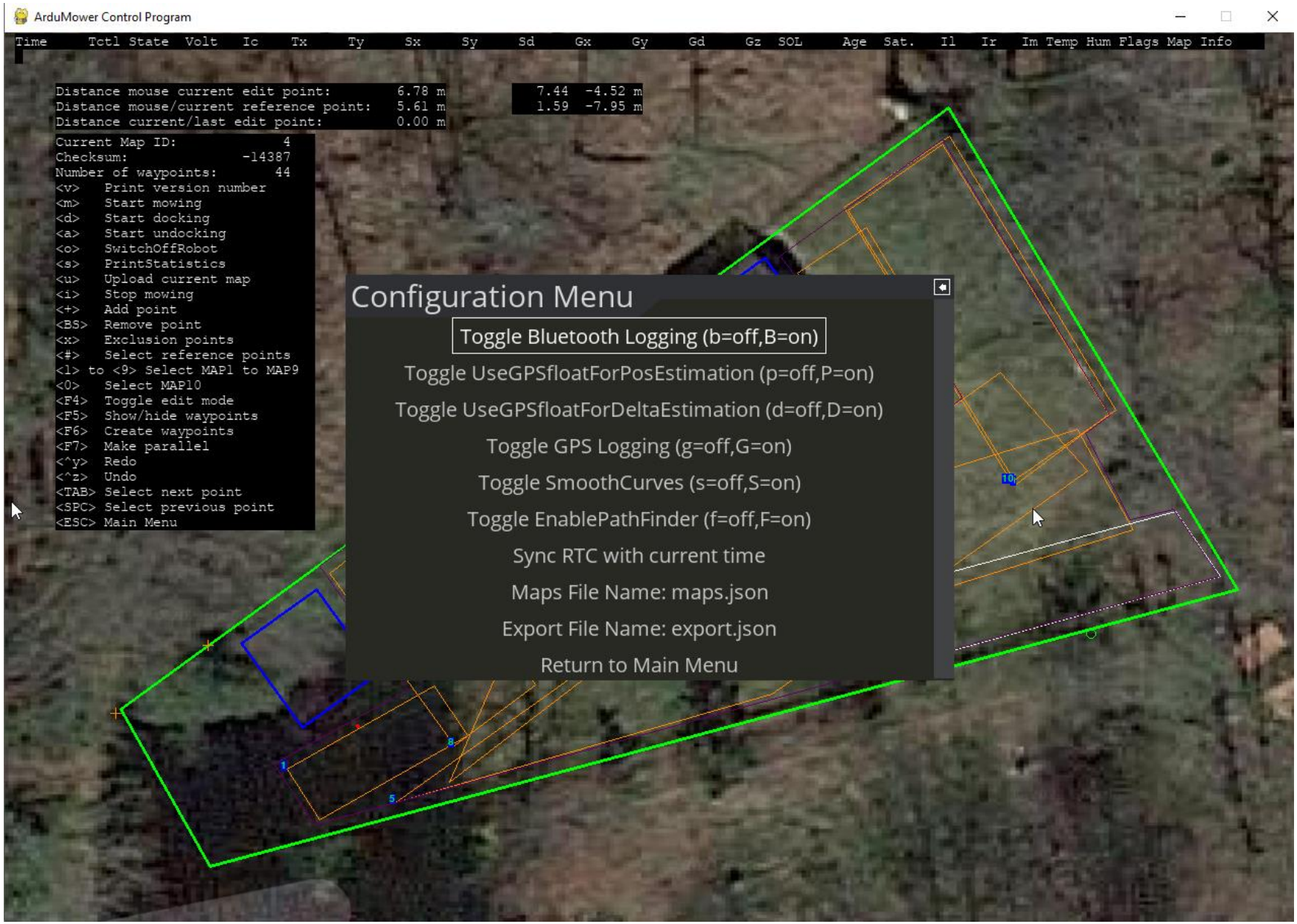
amcp – ArduMower Control Program

- Moonlight wird über das Python-Programm **amcp** gesteuert
- Aufruf im WinPython Cmd-Window: *amcp.bat*
- **amcp** kann entweder über Tastatur-Kürzel oder über Menü gesteuert werden
- Das Menü wird mit ESC ein- und ausgeschaltet.
- Die Tastatur-Kürzel sind nur verfügbar, wenn das Menü ausgeschaltet ist

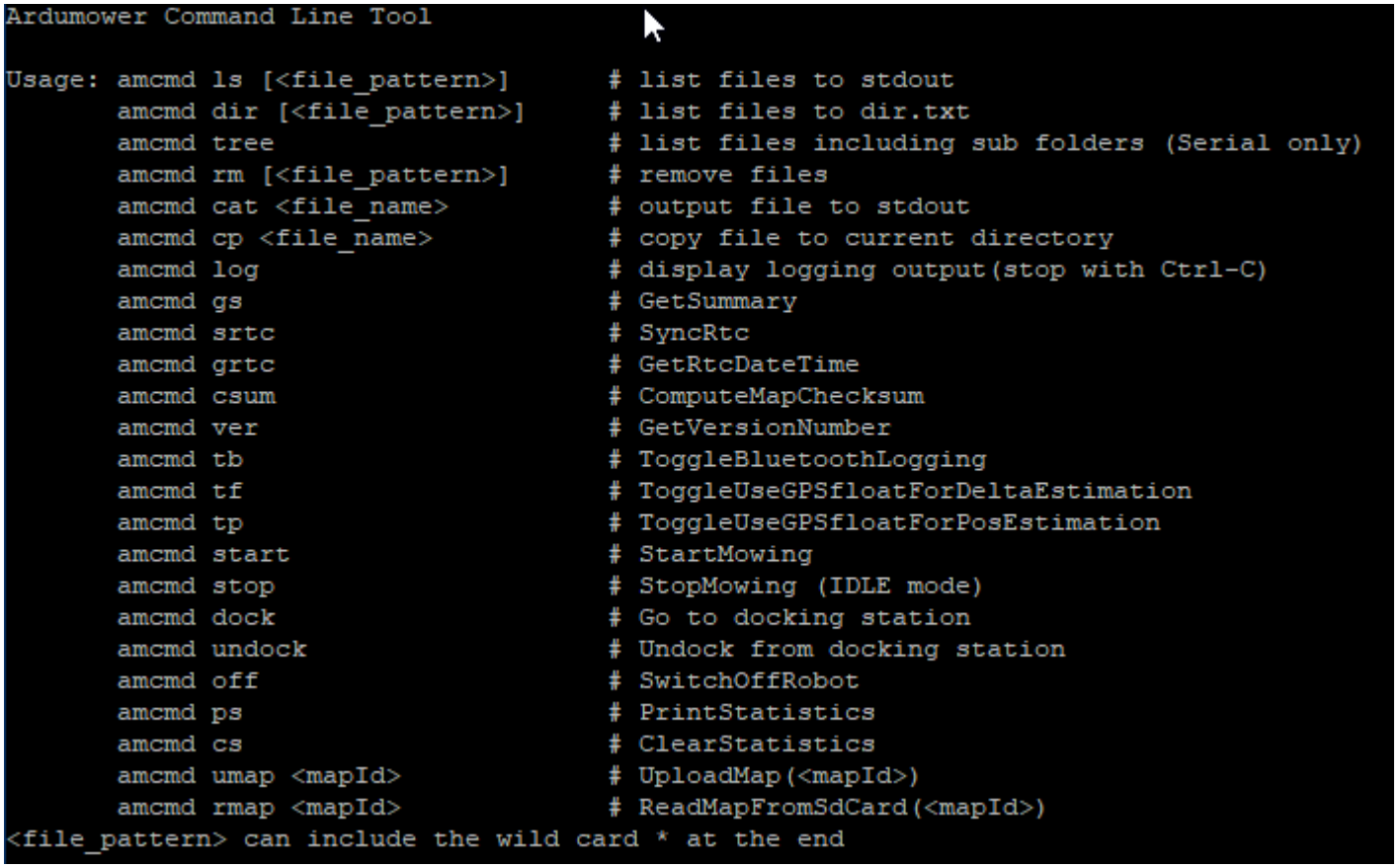
amcp – Main Menu



amcp – Configuration Menu



amcmd – Ardumower Command Line Tool



Fix Mode

- **No fix**: either there is not enough data to compute a navigation solution, or the computed solution is outside of the acceptable error criteria
- **3D**: a position solution has been achieved with at least four satellites as compared with a 2D solution with only 3 satellites and altitude/vertical metrics locked to a preset value.
- **DGNSS**: corrections are provided from a source that measures the differences between what pseudorange (signal time-of-flight approximately indicates distance using speed of light through a medium) values for each satellite should be at the precisely surveyed reference station compared to what is reported by its navigation computation. The differences are generally caused by time-varying ionospheric perturbations in the path from satellite to the reference station and the nearby rover. If the rover and reference are too far apart, the separate signal paths may have different perturbations and the correction data are less effective.
- **Float and fixed**: in precise navigation, the receiver tries to lock on to carrier phase (within the 19 cm wavelength) to resolve the exact number of wavelengths and fractions to each satellite. If this exactitude is achieved, the status is "fixed" since phase markers can be locked. if precise wavelengths and fraction are not resolved, ambiguity persists, and the status is "float" since the carrier phase markers are not locked but "floating around" in the mathematical computation. The difference is mm of position accuracy with fixed compared with cm of accuracy with float.

Main Program Loop

- always
  - robotDriver.run()
  - buzzer.run()
  - stopButton.run()
  - battery.run()
  - batteryDriver.run()
  - motorDriver.run()
  - motor.run()
  - sonar.run()
  - maps.run()
  - rcmodel.run()
  - fixDisplay.run()
  - gps.run()
  - calcStats()
  - processComm()
  - outputConsole()
  - watchdogReset()
- every 60 sec
  - temperature & humidity
- every 5 sec
  - saveState()
- every 150 ms
  - IMU handling
- every 20 ms
  - computeRobotState()
  - trackline()
  - battery & charging
  - on/off button

```
while (true) {
    robotDriver.run();
    buzzer.run();
    stopButton.run();
    battery.run();
    batteryDriver.run();
    motorDriver.run();
    motor.run();
    sonar.run();
    maps.run();
    rcmodel.run();
    fixDisplay.run();

    if (millis() >= nextSaveTime) { nextSaveTime = millis() + 5000;   saveState();  }
    if (millis() > nextTempTime)  { nextTempTime = millis() + 60000;  handleTemp(); }
    if (millis() > nextImuTime)   { nextImuTime = millis() + 150;     handleIMU();  }

    gps.run();
    calcStats();

    if (millis() >= nextControlTime) {
      nextControlTime = millis() + 20;
      computeRobotState();
      handleChargerAndDockingStation();

      if (!imuIsCalibrating) {
        if (stateOp == OP_UNDOCK) handleUndocking();
        else if ((stateOp == OP_MOW) ||  (stateOp == OP_DOCK)) {
          if (driveReverseStopTime > 0) HandleDriveReverseAfterObstacleDetection();
          else HandleLineTracking();
          HandleBattery();
          HandleStopButton();
          battery.resetIdle();
        }
        else if (stateOp == OP_CHARGE) HandleCharging();
      } // !imuIsCalibrating
      if (stopButton.LongKeyPress()) HandleSwitchoff();
    }
    processComm();
    outputConsole();
    watchdogReset();
}
```