# Pololu 3pi+ 32u4 Referenz

## Bibliothek

Bibliothek einbinden und mit `using namspace` einfach verfügbar machen:

```
#include <Pololu3piPlus32U4.h>
using namespace Pololu3piPlus32U4;
```

## LEDs (red, green, yellow)

Verschiedene LEDs schalten:

```
ledRed(1);    // LED einschalten
ledGreen(0);  // LED ausschalten
ledYellow(0); // LED ausschalten
```

## Batterie-Spannung

Gibt die aktuelle Batteriespannung zurück:

```
int spannung = readBatteryMillivolts();
```

## Taster

Objekt für den Taster erzeugen:

```
ButtonA buttonA;
```

Zustand des Tasters B abfragen:

```
if (buttonB.isPressed()){
 …
}
```

Pausiert das Pogramm, bis Taster B gedrückt wurde:

```
buttonB.waitForButton();
```

## Motoren

Objekt für die Motoren erzeugen:

```
Motors motors;
```

Geschwindigkeit einstellen (max: –400 … 400):

```
motors.setSpeeds(200, -200);
```

## OLED-Display

Das OLED-Objekt erzeugen

```
OLED display;
```

Konfiguriert das Layout des Displays:

```
display.setLayout8x2()  // 8x2 Zeichen
display.setLayout11x4() // 11x4 Zeichen
display.setLayout21x8() // 21x8 Zeichen
```

Löscht den Inhalt des Displays:

```
display.clear();
```

Positioniert den Cursor (X: Zeichen, Y: Zeile):

```
// 1.Zeichen, 2. Zeile
display.gotoXY(0,1);
```

Text oder Daten auf dem Display anzeigen:

```
display.print("Text"); // Text
display.print(wert); // Daten
```

Zum Löschen von Buchstaben Leerzeichen nutzen:

```
display.print("     "); // Leerzeichen
```

## Liniensensoren initialisieren

Erzeugen des Objektes:

```
LineSensors lineSensors;
```

Kalibrieren der Sensoren (messen und ggf. aktuell kleinsten oder größten Wert neu festlegen):

```
lineSensors.calibrate();
```

## Liniensensoren auslesen

Ein Array wird zum Speichern der Sensorwerte benötigt:

```
unsigned int senValues[5];
```

Die Sensorwerte können unkalibriert …

```
lineSensors.read(senValues);
```

… oder kalibriert ausgelesen werden:

```
lineSensors.readCalibrated(senValues);
```

Die aktuelle Sensorwerte sind im Array abgelegt:

```
int leftSensor = senValues[0];
```

Position des Zumos über der Line ausgeben (die Sensorwerte im Array werden auch aktualisiert):

```
int pos =
lineSensors.readLineBlack(senValues);
int pos =
lineSensors.readLineWhite(senValues);
```

## Rad-Encoder (Odometry)

Das Objekt für die Encoder erzeugen:

```
Encoders encoders;
```

Gibt die aufsummierten Encoder-Impulse zurück:

```
int tLeft = encoders.getCountsLeft();
int tRight = encoders.getCountsRight();
```

Gibt ebenfalls die Anzahl der Impulse zurück, setzt den Wert anschließend aber auf null:

```
int tLeft =
  encoders.getCountsAndResetLeft();
int tRight =
  encoders.getCountsAndResetRight();
```

## Kontaktsensoren (Bump Sensors)

Das Objekt für die Kontaktsensoren erzeugen:

```
BumpSensors bumpSensors;
```

Vor dem Verwenden müssen die Sensoren kalibriert werden:

```
bumpSensors.calibrate();
```

Die Sensoren müssen *regelmäßig* mit `bumpSensors.read()` ausgelesen werden, um anschließend den Zustand zu überprüfen:

```
void loop(){
  bumpSensors.read();
  if (bumpSensors.leftIsPressed()){
    // wenn links gedrückt ...
  }
  if (bumpSensors.rightIsPressed()){
   // wenn rechts gedrückt ...
  }
}
```

## Gyroskop

Folgenden Bibliotheken müssen zusätzlich eingebunden werden

```
#include <Wire.h>
#include <Pololu3piPlus32U4IMU.h>
using namespace Pololu3piPlus32U4;
```

Das Objekt für das Gyroskop erzeugen:

```
IMU imu;
```

Konfiguration des Gyroskops mit:

```
Wire.begin();
imu.init();
imu.configureForTurnSensing();
```

Neue Messwerte einlesen ...

```
imu.readGyro();
```

und anschließend auf die Werte der einzelnen Achsen zugreifen:

```
int gyroX = imu.g.x;
int gyroY = imu.g.y;
int gyroZ = imu.g.z;
```