```
#include <PID_v1.h> //Install PID library by Brett Beauregard v1.2.0 from Manage Libraries Section

double Required_RPM ; // will be the desired rpm value
double Current_RPM;
double Output_RPM;

double Kp=0.5, Ki=0.01, Kd=70; //PID parameters
float Total_Distance_Travelled = 0;
float Total_Revolutions = 0;
float Remaining_Distance = 1050;
float Remaining_Time = 0;
// Motor encoder output pulse per rotation (change as required)
#define ENC_COUNT_REV 374

// Encoder output to Arduino Interrupt pin
#define ENC_IN 3

// Hbridge connections to pin 10 and 11
#define Motor1_Out_1 10
#define Motor1_Out_2 11
#define Motor2_Out_1 5
#define Motor2_Out_2 6

// Pulse count from encoder
volatile long encoderValue = 0;

//create PID instance
PID myPID(&Current_RPM, &Output_RPM, &Required_RPM, Kp, Ki, Kd, DIRECT);
```

```
void setup()
{
  // Set encoder as input with internal pullup
  pinMode(ENC_IN, INPUT_PULLUP);

  // Set Motor connections as outputs
  pinMode(Motor1_Out_1, OUTPUT);
  pinMode(Motor1_Out_2, OUTPUT);
  pinMode(Motor2_Out_1, OUTPUT);
  pinMode(Motor2_Out_2, OUTPUT);

  Output_RPM = 1500;
  analogWrite(Motor1_Out_1, 255); //a value of 255 corresponds to 1500 rpm
  analogWrite(Motor1_Out_2, 0); //Forward Motoring
  analogWrite(Motor2_Out_1, 255);
  analogWrite(Motor2_Out_2,0);

  //Turn the PID on
  myPID.SetMode(AUTOMATIC);
  //Adjust PID values
  myPID.SetTunings(Kp, Ki, Kd);

  // Attach interrupt
  attachInterrupt(digitalPinToInterrupt(ENC_IN), updateEncoder, RISING);
}

void loop()
{
  Current_RPM = (encoderValue * 60 / ENC_COUNT_REV);
  Total_Revolutions = encoderValue / ENC_COUNT_REV;


  Current_RPM = map(Current_RPM, 0, 1500, 0, 255);  // photo senor is set on analog pin 5/
  Total_Distance_Travelled = Total_Revolutions * 44; //44cm is the distance covered per revolution
  Remaining_Distance = 1050 - Total_Distance_Travelled;
  Remaining_Time = 1300 - millis();
  Required_RPM = (Remaining_Distance / (Remaining_Time * 44))* 60;
  //PID calculation
  myPID.Compute();
  if (Required_RPM>0)
  {
    analogWrite(Motor1_Out_1, Output_RPM);
    analogWrite(Motor1_Out_2, 0); //forward motoring
    analogWrite(Motor2_Out_1,  Output_RPM);
    analogWrite(Motor2_Out_2,0); //forward motoring
  }
  else {
    analogWrite(Motor1_Out_1, 0);
    analogWrite(Motor1_Out_2, Output_RPM); //reverse motoring
    analogWrite(Motor2_Out_1, 0);
    analogWrite(Motor2_Out_2,Output_RPM); //reverse motoring
  }



}


void updateEncoder()
{
  // Increment value for each pulse from encoder
  encoderValue++;
```

```
}
```