Those of you who will get their driver's license soon will have to deal with one huge nightmare: parallel parking! You move your car back and forth, but in the end, you're still not in the right spot and you never move your eyes away from the mirrors, because you fear you might bump into the car behind you.

Thank goodness that most modern cars come with an integrated park distance control system, which keeps you from bumping into other cars with the help of loud, increasingly fast beeping.

The answer is hidden: “invisible light”! Or rather infrared light, which you know from infrared remote controls. Light in the infrared spectrum is invisible to the human eye. It is sent by a sensor, reflected on a surface, and then registered by a receiver. You will learn how to measure distance and how to create warning signals with the help of the Arduino microcontroller. That way, you will hopefully prevent any bumps in the buffer-bar of our test car ;-)
The infrared distance measuring sensor “Sharp GP2Y0A21YK” is a complex device with an integrated connection that can measure distances between 10 and 80cm pretty accurately.

The sensor has two “eyes”, a sender and a receiver. The sender sends an infrared light, which is reflected by a surface and then hits the receiver – always in a different angle depending on the distance. With this information the device can calculate the distance with a particular mathematical method. The sketch on the left shows you how it’s done.

The acoustic Piezo-signaler is a so called “buzzer” or “beeper”, a small device that is controlled electronically to play a particular sound – so it is an acoustic display. They are used everywhere that warning signals or notification in the form of sound is needed – for example in smoke detectors, in microwaves or as beeping in a park distance control system.

Now we can start the construction

These steps will help you with your wiring (the patch panels will not be needed for that)

1. **With the help of the blue cable**, connect the minus pole (the shorter leg!) of the signaler with one of the GND pins of the Arduino (at the digital pins).
2. **With the help of the red cable**, connect the plus pole (the longer leg!) with one of the digital pins [e.g. 8], which you will use to control the output.  
   **[Note: Pins 0 and 1 cannot be used for this.]**
3. The IR-distance-senor is directly connected to the Arduino. Use the black cable to connect it to GND (at the analog pins), the red cable to connect it to 5V and the white cable to connect it to one of the analog pins [e.g. A0], which you will use as the input pin to measure the distance.
Analog Pins

So far, you’ve only got to know digital pins, which you used to connect an LED. These pins can only read in and out binary values – so like on/off, 0/1 or the values that you already know HIGH and LOW for high or low voltage.

A distance sensor on the other hand can receive more than just two values, it measures many different interim values in the minimum and maximum value range. This is detected by the analog pins on the Arduino (A0 to A5), which is used to measure whole regions.

[Note: Your wiring can look different from the picture on the next slide (for example the choice of the pins). That’s absolutely fine, there is not only one solution!]
Informatics Enlightened
Station 2 – Park distance control system

Manual distance measurement

First, you will read in the data that the infrared sensor measures with the help of a little program and then display them in the serial monitor. So your first assignment is to use the IR-sensor to measure the distances and fill in the different centimeter values into the table on page 7. You have to do that so that you know which values match the different centimeter distances.

The analog input pins of the Arduino will always give you a value between 0 and 1023, so we need to be able to translate that to centimeters. But that’s actually not that simple, there is no quick mathematical formula to do this conversion (I mean, the graph does look pretty weird).

![Graph showing output values of the IR distance measuring sensor between 10 and 80cm](image)

**Fig. 5: Output values of the IR distance measuring sensor between 10 and 80cm**

[Note: You will later get a formula, which you can use to compare the data that you’ve filled into the table!]

In this first step, you will learn how to...

- ... read in the sensor values using an analog pin and
- ... display those values on the serial monitor.

Your to-do’s

1. Open a new Arduino sketch and save it under a reasonable name.
2. Create the default structure, which you know from the introduction.
3. Before you can start working on setup() and loop(), you have to define (just as in the introduction) an int variable for the analog input pin and assign it the correct number of the pin.

[Note: Even though the pins are called A0, A1, etc., you only use the number in your sketch!]
4. Now, you have to declare a second variable, where you will save the sensor data, so that you can use them any time you need. Don’t forget to use appropriate names!

5. Let’s do `setup()`:
   a. How do you start the transmission of the serial monitor?

6. Now, let’s do `loop()`:
   a. The most important step is the reading of the sensor values. Different from setting the voltage at a digital pin to HIGH or LOW, you need to use this command:

   ```
   <variableName> = analogRead(<Pin-name>);
   ```

   to read in the sensor values. So assign the values to the variable from step 4 with the help of the “=”-symbol and choose the correct pin.

   ```
   ______________ = ______________( ______________);
   ```

   b. Now, we need the display on the serial monitor, which will be initiated with `Serial`. Include a short description of what should be displayed. Don’t forget to use inverted commas.

   c. Add a second output, which should include the variable of the sensor values in the brackets. This time don’t use inverted comma, because you only need those for texts! But in this case, we want to display the sensor values, which are saved in the variable.

   d. Add a `delay(<time>)`; at the end of the command, which updates the values every one or two seconds. Otherwise, the values will be displayed lightning fast in the serial monitor.

   ```
   delay(___________);
   ```

7. It’s time to test your sketch! Connect the Arduino again, run the program and watch the sensor values.

You will soon notice that the distance values are not between 10 and 80cm.

```
min = ___________ max = ___________
```

It’s not enough just knowing the minimum and maximum values, we want to know how those values translate to the distances between 10 and 80cm. For that, you need a piece of paper and
Informatics Enlightened

Station 2 – Park distance control system

a **ruler** to determine which values are displayed for 10, 20, 30 cm, etc. Fill in these values in the table!

[**Hint:** The IR sensor measures most accurately, if you use a white piece of paper to reflect the infrared light!]

<table>
<thead>
<tr>
<th>distance</th>
<th>10 cm</th>
<th>20 cm</th>
<th>30 cm</th>
<th>40 cm</th>
<th>50 cm</th>
<th>60 cm</th>
<th>70 cm</th>
<th>80 cm</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>values</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Don’t get frustrated! Even if you hold the sensor very steady, it still might give you **different values** for the same distance. Just use the **average** of the values you got.

A simple warning signal

Let’s get back to **programming**. For now, the sensor measures the distances, but we don’t have a warning signal yet. For that, you have to use the second device, the **Piezo signaler**.

**In the next step, you will learn how to...**

- ... include and control the **Piezo signaler** in your sketch and how to create an **acoustic signal** relative to the sensor values.

**Your to-do’s**

1. Save your Arduino sketch under a different name (“**Save under**”). From now on, you will expand, change and adjust your program again and again.
2. Come up with a new **int** variable for the Piezo signaler and assign it the **digital pin** which you used to connect it to the Arduino.
3. **setup()**: Here, you have to define the Piezo signaler as **output** (since it creates a sound) – just like you did for the LED in the introduction.

```cpp
pinMode(________________________, ___________________________);
```
4. In `loop()`, you need the good, old `if-else` command you know from the introduction.

**if-else-command**

If you want to define in your sketch that something should happen given a certain condition – for instance creating a beeping sound, if the distance is too small – you can use the so called **if-else-command**. This is how it is used:

```c
if(<condition>) {
  <command>;
  ...
} else {
  <command>;
  ...
}
```

If the condition is **true**, the first command is executed, if it is false, the second command is executed. You usually use **mathematical comparisons** (`>`, `<`, `>=`, `<=`, `==`) for such a condition. There is also the **logical operator && [and]** to check whether two conditions are true at the same time:

```c
if(<condition> & & <condition>)
```

a. Now we have to use such an if-else-command. Set up a structure for that, just like the one in the box.

b. You want the Piezo signaler to beep – create a **continuous sound** – if the distance to the infrared sensor is too small (here **under 40 cm**). So, which variable do you need to come up with the correct condition? Have another look at the table: Which boundary value do you have to compare it to?

c. If your condition is true, so the distance is too small, a continuous sound should be created. For that, you need to set the **digital output pin** to **HIGH** (just like you did for the LED).

d. In any other case, there shouldn’t be any sound. Think about what command you have to include in your if-else-command, so that the Piezo signaler is “switched off” again. Write down your thoughts in this schema:

```c
if(__________________________<condition>=__________________________) {
  __________________________;
} else {
  __________________________;
}
```

5. **Done!** Transfer the sketch to the Arduino and test it. Are you happy with the distance you set? Do you hear the sound too early/too late?
What now?!

Do you have some other ideas or suggestions for improvements? Discuss them in your group and write them down below! In the next two sections, you will keep on improving your park distance control system.

Keep on experimenting and programming! We wish you a lot of fun!!! 😊

Notes:

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

„Beep, Beep!“

Or not.. So far, your control system makes one continuous sound, if the distance is too small. But a driver wants to know how much space he has left, if he’s already too close or if he has nothing to worry about!

All that can also be done with the help of the Piezo signaler, it can just beep again and again.

In this improvement, you will learn how to...

- ... set up distance intervals with the help of the sensor values and
- ... create a beeping sound using `digitalWrite(<Pin-name>, <CONDITION>);` and `delay(<time>);`

Your to-do’s

1. You’ve got everything set up, so you can get started right away by saving our old sketch under a new name!
2. You will only need to change the `loop()`, or rather just the `if-else`-command:
   a. Have a look at your table on page 6 and think about an appropriate interval of sensor values and centimeters in which no sound is necessary and two others in which there should be beeping of different speeds.
b. Now you just have to adjust your if-else-command a little bit, by adding these intervals you’ve just come up with. You have to change your one if-else-command (which can only distinguish between two intervals) to three if-commands.

[Hint: Use the mathematical comparisons and especially the logical & [and] and change the speed of the beeping with the help of delays.]

For each of the three intervals you need a separate if-command like this:

```c
if(________________________ ___ _____) {
    digitalWrite(_______________, _______________);
    delay(______________);
    digitalWrite(_______________, _______________);
}
if(_______________ ___ _____ & & _________________ ___ _____) {
    digitalWrite(_______________, _______________);
    delay(______________);
    digitalWrite(_______________, _______________);
}
if(________________________ ___ _____) {
    digitalWrite(_______________, _______________);
    delay(______________);
    digitalWrite(_______________, _______________);
}
```
3. Now, you’re ready for the **testing phase**! Transmit your sketch and test your improvements. Change the intervals if need be, until you’re happy with the result.

Done?!?

Congratulations! You did it! The test car can now be parked with ease just because of you 😊

If you like, there is another challenge for you...

Just ask your instructor for the bonus assignment!

---

**List of references:**

- **Fig. 1** – Source: Mozilla “Parking assist“ under license CC BY-SA 3.0 / altered original
- **Fig. 2, 3, 4.1, 5** – Source: InfoSphere
- **Fig. 4.2** – Source: Screenshots of the Fritzing-Software ([http://fritzing.org](http://fritzing.org))

All other graphics/icons – Source: InfoSphere