Lab 4

Servo Motors and Gears

Authors:

Alexandre Mesot, Jonas Lussi, Naveen Shamsudhin and Prof. Bradley J. Nelson

Multi-Scale Robotics Lab, Institute of Robotics and Intelligent Systems

Date: 2021

Version: 1.3

Summary: In this lab, we will introduce servo motors and gears by solving some simple taks. We will learn:

- How servo motors work and how to control them using the Arduino interface.
- How to increase/decrease the torque and run length of the servo motors using combinations of gears.
- How to design gear trains for specific applications and requirements

4.1 Background

There are many different types and divisions of electric motors: AC or DC motors, Stepper Motors, Rotary and Linear motors and many more. Servo motors (or servos in short) are linear or rotational motors with a very high positional precision. Any applications that need to move or rotate an object to a precise location can use servo motors. Such applications are manifold and range from RC planes and toy cars, to opening and closing DVD and Blu-ray Disc trays to operating grippers in all kinds of robots. Servos are also often used for in-line manufacturing, where they shine in highly repetitive, yet very precise work areas. Servos are generally fairly small and offer a very high torque compared to their size. The IRM lecture notes on **actuators** provide additional background information for this lab and feel free to do your own research if your interest has been piqued.

4.1.1 Servos, control and pulse width modulation (PWM)

In the housing of servos there are 4 components: A (DC or AC) motor, a gear assembly, a potentiometer, and a controlling circuit (Figure 4.1). The gear assembly reduces the rotational speed of the motor in order to increase its torque. The potentiometer is attached to the drive shaft and is used to determine the position of the servo. The control circuit reads the output of the potentiometer and compares it to the the command signal, which it receives from the user. The difference is processed in a feedback loop which outputs a signal for the motor to start rotating. This will start turning the potentiometer as well, thus reducing the difference between its output and the command signal. The loop is repeated until the difference is zero and the new position has been reached . This position measuring mechanism is the reason why servos usually can only rotate 180°(or any fixed angle). While the DC motor can rotate continuously, a potentiometer cannot. When the difference in position is large the servo will move as fast as possible to close the gap. As it gets closer to its desired position it will continually slow down so as to not overshoot the position.

Potentiometer: A potentiometer is a three-terminal variable voltage divider. It has a resistive element inside. At each end of the element one terminal is rigidly attached. The third, central terminal wipes over the resistive element when the knob is turned. The closer the wiper is to the end terminal, the less the resistance is. The further away, the greater it becomes.

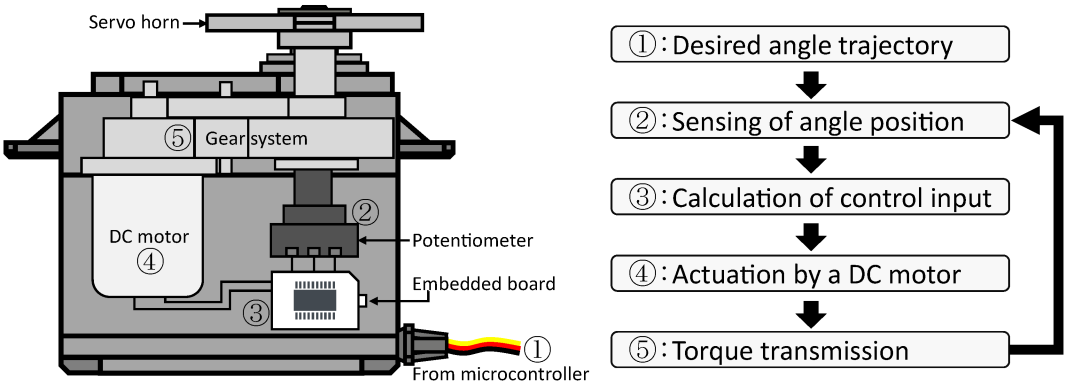

Figure 4.1: Schematic of a DC servo motor and sequence to move to a desired position.

Servos can usually rotate from 0°to 180°. The command signal to set a desired position is sent in the form of a PWM (Pulse Width Modulation) signal. This means that the position of the servo is encoded in the length of the pulse that is sent. Generally servo motors expect to receive a pulse every 20 ms (or at a frequency of 50 Hz), with the pulse width usually ranging from 1 ms (to encode the position of 0°) to 2 ms (to encode the position of 180°). The position scales linearly between the two extremes, meaning that to reach the 90°position a PWM of 1.5 ms is needed (Figure 4.2). Keep in mind that the servo needs some time to reach its position. Some servos come with an additional cable, which outputs its current position. This can be used in a feedback loop to perform a series of consecutive movements without having to estimate how long it will take for the servo to reach each position.

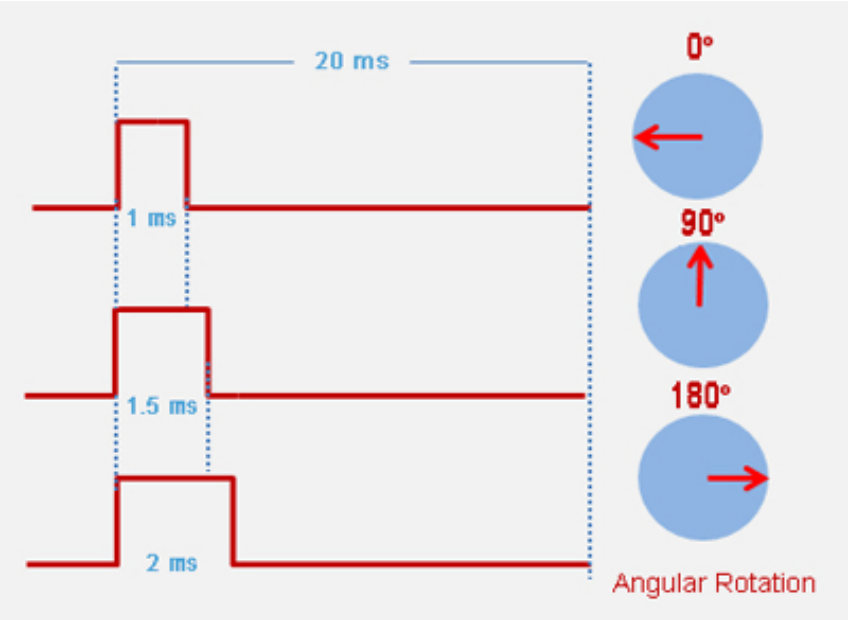

Figure 4.2: Pulse Width Modulation is used to encode the position of the servo in the length of a pulse.

Note that continuous rotation servo motors (CRS) which can rotate indefinitely in any direction also exist. Instead of having a very precise control over its position, CRS have a very precise control over speed and direction. The rotational speed is determined using a similar principle as with the normal servos. The potentiometer is replaced by a pair of identical resistors that create a fixed voltage divider which acts as if the potentiometer is always at its

mid point (let's say 90°). When you send a PWM signal for the 180° position the CRS will start moving fast, in order to reduce the large gap. However as there is no potentiometer the gap will remain the same and the speed will remain constant. If you now send a PWM signal for the 85° position the CRS will start turning in the other direction, as well as turning very slowly, as the gap is much smaller than before. This way the control circuit can be the same in both types of servos and the only thing that has to be changed in manufacturing is to replace the potentiometer with a pair of resistors. Some "normal" servos can be converted into CRS. This is usually labeled as **360°modifiable**, which can be misleading, as the modifying process is fairly tedious. The servo has to be opened, the potentiometer removed and fixed resistors soldered in its place and a physical stop on one of the gears has to be cut away to allow for full rotations. For this reason the modification is only done when you do not want to spend additional money on a new servo. This means that it is important to always define your requirements before you decide to buy a servo motor or a CRS.

Note: Whenever "servo motor" is mentioned during this lab, you can assume that it is a non-continuous servo motor. Continuous rotation servos will be specifically named "CRS" in order to avoid any confusion.

4.1.2 Torque and Gears

As you have seen in the lecture, electric motors have a linear relationship between rotation speed and torque. The operational rotation speed at which the output power is maximal is at $\omega = 0.5 * \omega\_max$. Typical rotation speeds in motors are in the 1000s of rpm, in order to maintain this ideal power output. Applications however need more reasonable rotation speeds, as nobody wants to run through a revolving door at 30 turns per second. Combinations of gearheads are thus very important in robotics and almost any kind of motor applications because they reduce the rpm and at the same time increase the load that the motors can handle. This can be seen by looking at the relationship between power, angular velocity and torque: $P = \omega * T$. The ideal power output of the motor is kept constant. By using gear trains the rotational speed is decreased which then leads to an increased torque. It is the same principle that is used in bicycles and cars when gears are shifted in order to maintain the engine (or your legs) in the regime where most power can be generated.

In this lab you will be working with spur gearheads in order to increase the torque of a servo motor, as well as to increase its range of motion. Real life applications have additional requirements that will make the choice of the correct gears more elaborate. Accuracy, backlash, efficiency, lifetime, load, noise reduction, inertia matching and torque multiplication often have all to be considered in the decision-making process. Here are some important relationships:

Gear Ratio: The gear ratio is the ratio between the rotational speeds of two connecting gears. As each gear has a different diameter, each axis rotates at a different speed when engaged. The gear ratio is calculated by dividing the input speed by the output speed ($r = \omega_{in}/\omega_{out}$), or by dividing the number of teeth of the output gear, by the number of teeth of the input gear ($r = Z_{out}/Z_{in}$).

Torque: The amount of torque that is transmitted is also proportional to the gear ratio. The output torque is calculated by multiplying the input torque with the gear ratio ($T_{out} = T_{in} * r$)

Compound gears: When two or more gears are fixed together on the same axis of rotation they are called a compound gear (Figure 4.3). Compound gears rotate at the same angular velocity, even though they usually have a different amount of teeth. This allows for a much greater increase or decrease of speed from the driving to the driven shaft, compared to a classical gear train. The gear ratio for the whole compound gear train, assuming gear A is the driving gear, is: $r = Z_{GearB}/Z_{GearA} * Z_{GearD}/Z_{GearC}$. Follow this link if you want more information about compound gears:

http://www.technologystudent.com/gears1/gears3.htm

4.1.3 Adafruit Motor Shield/Wing

Arduino boards are very useful for general electronic applications, however some specific applications need additional features. Shields are pieces of hardware that you can mount on Arduino or Arduino-compatible boards in order to add these additional features. Popular examples are Ethernet Shields to add internet capabilities, GSM/GPRS Shields to add mobile data connectivity, LCD Shields to add an external display and Bluetooth Shields to

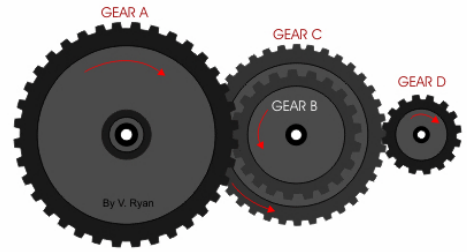

Figure 4.3: Gears B and C are combined to form a compound gear.

connect the board with your phone. In the case of robotics, Arduino boards like the Featherboard quickly reach their limits, as its pins cannot supply enough power to drive multiple motors. Motor shields can support that additional load as they have their individual power supplies, without altering the simple and useful Arduino interface. If no power supply is attached to the Motor shield the power comes directly from the 5V power output of the Featherboard. The maximum current that the Featherboard can supply is 500 mA, which is just enough to drive a single small Servo. It is highly recommended to connect a 5V or 6V power supply to the motor shield, if additional Servo motors are added, as you might fry the board. Some motor shields can be stacked to theoretically drive hundreds of servos and motors. The motor shield that we will be using in this lab is an Adafruit 8-Channel PWM Servo FeatherWing with an optional power supply of 5V or 6V and specific connectors for servo motors. We will be powering the motor shield via the USB hub that we are already using to safely connect the Featherboard to your laptops. As you can see in Figure 4.4 the motor shield is plugged directly into the pins of the Featherboard. All of the Featherboard pins are redirected to the top of the motor shield, where they can be used to connect additional shields or devices. To access GPIO pins during the lab please plug your wires into the breadboard instead of the "holes" in the motor shield. 4.4).

Check out the following link if you want to know more about the Adafruit Servo Wing:

https://learn.adafruit.com/adafruit-8-channel-pwm-or-servo-featherwing

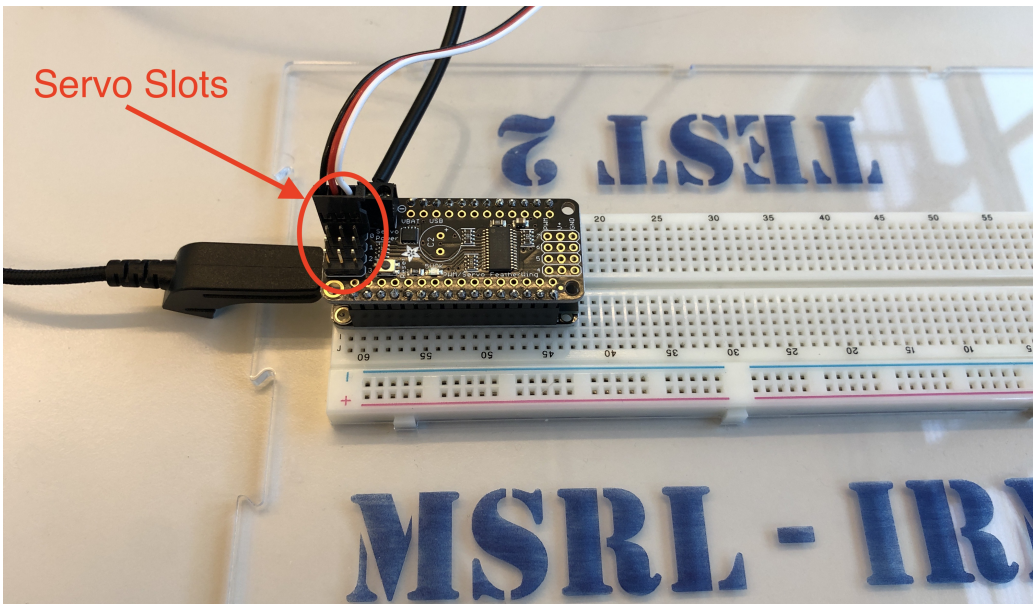

Figure 4.4: Featherboard with the Adafruit Servo Wing and a servo plugged in to the servo slot/channel 0.

4.2 Prelab Procedure

Note: The Prelab quiz on Moodle must be done before reporting to the lab. The submission deadlines are on Moodle. Late submissions will not be corrected. Each group member has to complete the quiz on their own, however you are allowed to work with students in your group to solve the quiz (and we absolutely recommend you to do so). Please remember to submit your quiz when you have answered all of the questions.

In this lab we will be using the Futuba s3003, which is a common DC servo motor. It has 3 cables connected to it: Black (ground), Red (5V-6V power) and White (Signal). Read the background on servos and gears as well as the data sheet of the Futuba s3003 and answer the questions in the Moodle quiz:

Read through the entire lab procedure. The key idea of reading a data sheet is to get a feeling for pinpointing the information of interest and filtering out the rest. If you don't understand something, prepare questions for the lab assistants.

4.3 Lab procedure

Note: Questions marked with **Postlab Qx** have to be answered as part of **PostLab 04**.

Note: Electronic equipment and components can be damaged if input voltage requirements are not met! Please be careful and follow the instructions. If you detect any burning smell, inform the assistant immediately!

4.3.1 Equipment Handling - Very Important, Read Before Proceeding!

- Whenever you plug wires in or out make sure to only hold them on the hard plastic part close to the tip. If you directly pull on the wire you risk ripping it out or damaging it in some other way. Take especial care if you pull the Servo out of its socket on the motor shield. Make sure to grip both the shield and the black hard plastic end of the cables when you separate them.
- When working with the supplied plastic gears do not worry if there is a loud grinding sound. This is to be expected, as the setup does not have very fine tolerances.
- In order to prevent wires from being ripped out unintentionally keep the breadboard and the servo-box attached to the plexiglas baseplate with the Featherboard on it.
- Remember that the analog input pins of the Featherboard can manage a maximal input voltage of 3.3 V. If you feed more than 3.3 V to the input you will damage the Arduino board. This is very important, since we will be working with 5-6V voltages to power the Servo motor. Make sure you do not feed these higher voltages to the GPIOs!
- If you have any questions do not hesitate to ask an assistant.

4.3.2 Servo position control via potentiometer

The Adafruit motor shield that is provided in this lab can manage up to 8 servo motors at the same time. The 3 pins for power, ground and PWM signal are aligned for ease of use and labeled. You can choose which slot to use, just remember that you will have to address the correct one in the Arduino code. We recommend you use slot/channel 0. Make sure to plug the Servo in the right way (**black wire** -> GND, **red wire** -> V+, **white wire** -> PWM) The power and ground pins are connected to the power supply and to the ground of the motor shield respectively. The PWM pin slots are connected over I2C to the featherboard (See Figure 4.4 for reference). In this part of the lab you will learn to control the position of the servo using a potentiometer and some code on Arduino.

1. Put the skeleton files from Moodle in a new Lab04 folder on your VM.

2. Assemble the the circuit in figure 4.5. Check the featherboard pinouts on the desks to find the correct pins. On your tables you will find a bunch of cables and a rotary potentiometer, which you will need for this part of the lab. You can set the resistance of the potentiometer anywhere between 0 and the maximum resistance by turning its knob. Make absolutely sure that you plug it into the 3.3 V and not into the 5 V pin or you will break the featherboard.
3. Plug the servo motor wires into one of the servo slots of the motor shield.

   Hint: The PWM signal wire (white) goes on the pin that is labeled PWM.

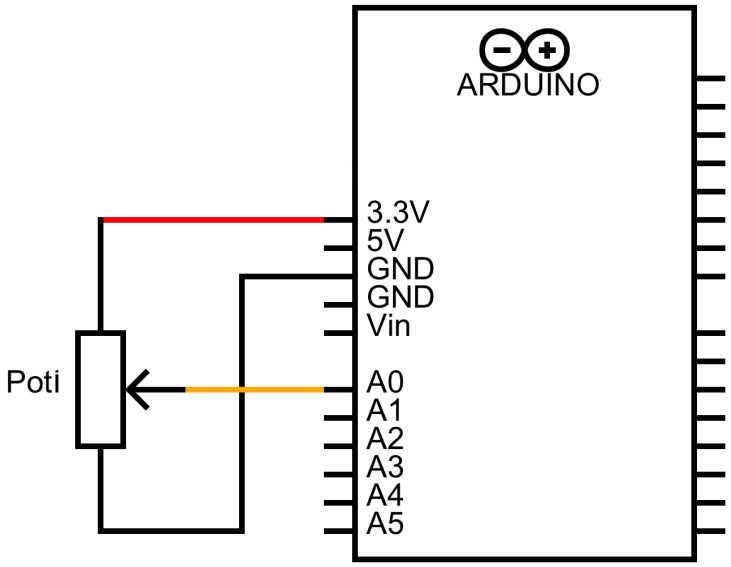

Figure 4.5: Potentiometer and Arduino circuit to set the position of a servo motor.

4. In order to be able to use the motor shield you will need to install a new library in the Arduino IDE. Go to **Sketch -> Include Library -> Manage Libraries...**. Search for the **Adafruit PWM Servo Driver Library**, install it and close the window. This library will include the commands and functions that you need to run a servo motor through the motor shield.
5. Open the *Servo_Poti.ino* file. Read the code to understand what the individual servo-related commands and functions are used for.

   Hint: If you want to familiarize yourself with the functions and commands of the Adafruit PWM Servo library go to **File -> Examples -> Adafruit PWM Servo Driver Library (At the very bottom of the list)** and open the **servo** example. It will show you all the relevant commands that you will be using during the lab. You can upload it to the featherboard to see what it does.

6. Using your knowledge from previous labs expand the code so that the μC reads the voltage over the potentiometer coming through ADC A0 (GPIO 26) and maps it to PWM pulse lengths for the servo in microseconds (0V triggers the 0° position, 3.3V triggers the 180° position). Use the newly mapped value to set the position of the servo. (**PostLab Q1**). Useful functions for this are:

   - analogRead(*PinNumber*)
   - map($x, fromLow, fromHigh, toLow, toHigh$)
   - pwm.writeMicroseconds(*servonum*, *microsec*)

   Hint: Output the signal you are receiving with analogRead() on the serial monitor to visualize the values and in order to find out what the resolution of the ADC is.

7. Now you should be able to set the position of the servo by turning the potentiometer on the breadboard.

4.3.3 Torque analysis

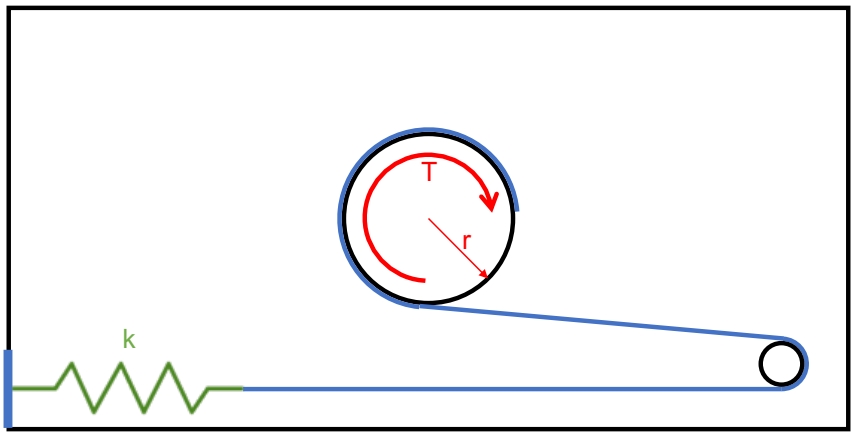

Figure 4.6: Torque analysis schematic.

1. Consider the situation described in Figure 4.6. A blue string is attached to a black disc with radius r and - passing over a return pully - to a green spring with spring constant k. The black disc is fastened to a servo motor with torque T. Using the values: $k = 500N/m$, $r = 0.02m$ as well as the torque of the servo that you can find on the datasheet and neglecting friction, how far should you be able to stretch the spring, before the servo stalls? (**PostLab Q2**)

   Hint: If an external power supply is feeding the motor shield, the same voltage is also supplied to the servo. The motor shield will automatically take its power from the 5V pin on the featherboard if there is no power supply. In this case the USB Hub is powering the motor shield. Inspect it to find the voltage it is supplying to the motor shield.

2. Build the same experiment as the one displayed on the sketch and on Figure 4.7 using the supplied materials. Securely attach the spring to one of the hooks and screw the spindle on the large central gear on top of the servo with 4 small slotted screws (if it is not there already). Pass the string through the second hook which will serve as a pulley and attach the hook on the string to one of the 3 screws on the side of the spindle. Use the circuit which you built in the previous part to deflect the spring as far as possible. Measure and write down the deflection and check if your calculations were correct. There will be deviations from the theoretical value. Calculate the efficiency of the system and name 1 cause for the losses/deviations. (**PostLab Q3**)

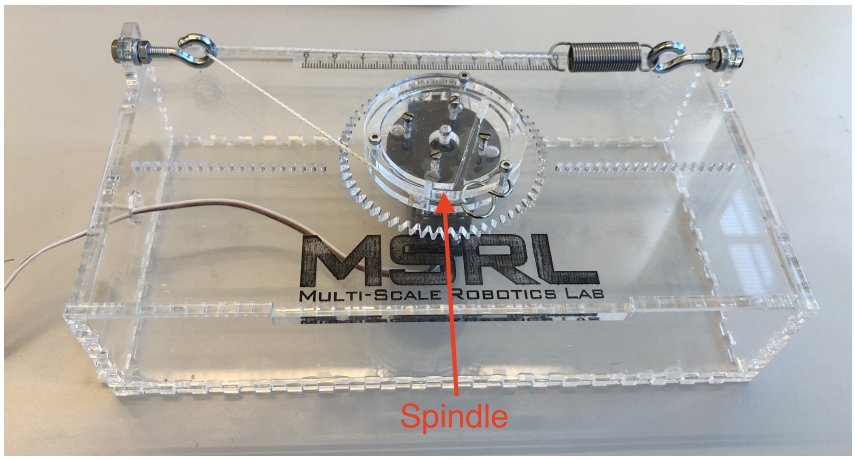

Figure 4.7: Example for the Torque Analysis starting setup.

3. Use a combination of gears to increase the force that you can apply on the spring, in order to get at least 1.5 times the deflection that you had before. The amount of teeth are engraved on the gears, so you do not have to count them. Determine the adequate gear multiplications that are needed and implement it on the servo

box. Use the pins, transparent spacer disks and hex nuts to place the gears on their correct spots as explained in Figures 4.8. Keep in mind that as you increase the torque you also reduce the maximum run length of the system. So you might end up with 4 times the force, which would easily pull the spring far enough, but having $\frac{1}{4}$ the run length you will not get far enough. Do not worry if you get slightly different values than your neighbouring teams, as the setups can and will vary. Write down the gear ratio that you used, as well as the calculations and thought process how you got to those values. Record a video of the longer displacement to show your assistants. Watch the video at the end of the lecture in order to see what it should look like. (**PostLab Q4**)

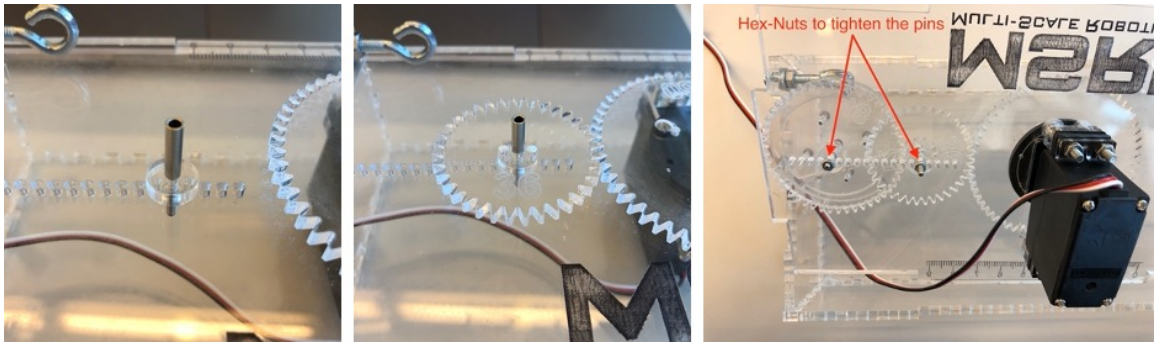

Figure 4.8: Put the pins on top of the spacers and then the gear on top of that to determine its placement on the box. Then use hex nuts to tighten the pins.

Hint: Remove the spindle from the big 60 teeth gear and attach it to the 48-teeth gear with the 4 threaded holes in it. Do not apply too much force when attaching the spindle to the gear, as you may break the threads in the plexiglas gear, as soon as you feel resistance stop tightening. When you have determined an adequate gear ratio use the red pin with the screw head to firmly attach it to the white box. The screw head will prevent the gear from being lifted off the pin (Figure 4.9). Use the small transparent spacer discs to position the gears at the correct height, you might need more than one spacer.

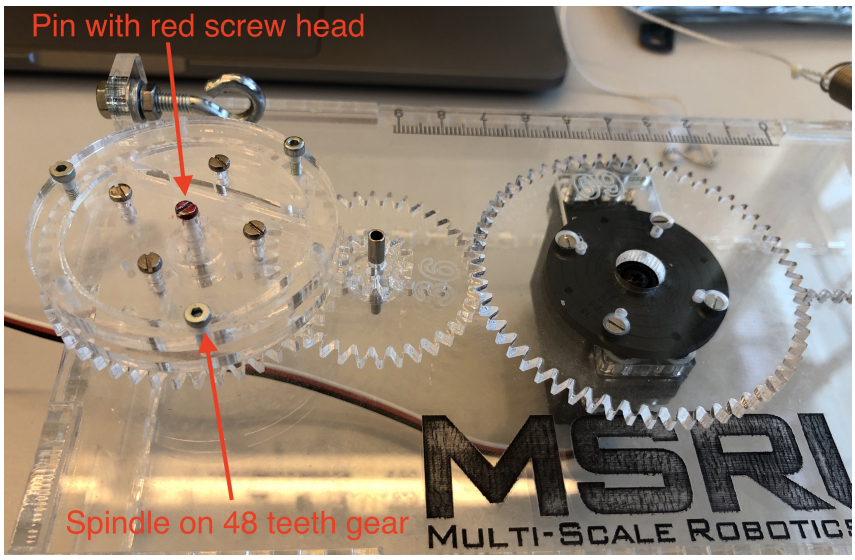

Figure 4.9: After attaching the spindle to the 48 teeth gear use the pin with the red screw head to fixate it. Tightening the pin with the screwdriver will prevent the spindle from being lifted off the pin.

4.3.4 Servo Timing Watch

The last task of this lab is to build an analogue timing watch, with a hand to display the seconds (0-60s) and a second hand to display the minutes (0-2.5min). Both will be driven simultaneously by the servo located in the

center. A schematic can be seen in Figure 4.10. As the range of motion of the servo is limited to 180° your task is to find a combination of gears so that the timer can run for a full 2.5 minutes by using up the 180° range of motion of the servo. Additionally you will complete the skeleton code in order to control the timer using the serial port. The idea is that you can set the amount of seconds you want your timer to run on a terminal window and then, every second a signal is sent to the Arduino to advance the watch hands by one second. Watch the video at the end of the lecture in order to see what it should look like.

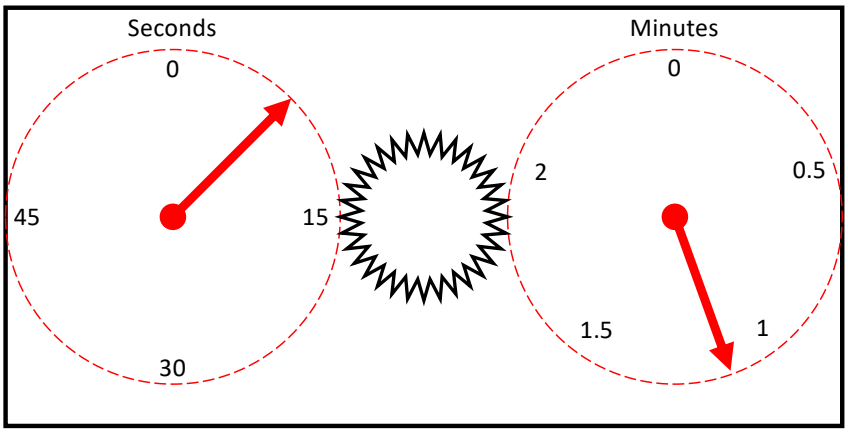

Figure 4.10: Schematic of the timing watch.

1. Calculate the gear ratios for both hands that are necessary for the watch to run for 2.5 minutes by rotating the servo motor 180°. (**PostLab Q5**)

   Hint: 1 minute represents a full rotation of the seconds watch-hand and 2.5 minutes represent a full rotation of the minutes watch-hand.

2. Calculate the theoretical resulting torque at both of the watch-hands. (**PostLab Q6**)

3. Remove the gears, pins, string and spring from the previous exercise and build the clock using combinations of the gears that are provided. Write down the combinations that you have used. (**PostLab Q7**)

   Hint: The watch hands are the red, pointy pieces with two pins and a hole in the middle. Pin down the watch dials under the spacer discs to prevent them from moving around when the gears move. Align the watch hands with the "0" position of the dials.

4. Open the *Servo_Timer.ino* file in the μC folder as well as the *timer.c* file in the μP folder.

5. Look at the function *delay*() at the top of the *timer.c* file. It is used to precisely time 1 second before sending the command to move to the Arduino board. Explain how it works. (**PostLab Q8**)

   Hint: The function *clock*() from the library *time.h* returns the amount of microseconds elapsed since the program was started.

6. Expand the skeleton code in the *timer.c* file so that when the program is executed it has the following functionalities: (**PostLab Q9**)

   (a) At startup: ask the user how long you want the watch to run (between 0 and 150 seconds) or if you want to quit the program.

   (b) If a sensible number has been entered send a signal (character "a" for instance) to the featherboard to reset the servo to its starting position. After this send a different signal (a different character, for instance "b") to the featherboard precisely every second through the serial port to tell it to advance the timer by 1 second. Repeat this until the set amount of time is reached.

   (c) When the timer is done return to the startup query, where you decide whether to continue with another timer or to quit the program.

7. Expand the skeleton code in the *Servo_Timer.ino* file and upload it to the Arduino so that it has the following behaviour: (**PostLab Q10**)

(a) Calculate the angle that the servo has to move in order for the watch hands to advance by 1 second on their respective time dials each time an input is received. Note that the previously used $servo.write(angle)$ command only works with *integers*. In order to have a higher resolution use the command $servo.writeMicroseconds(lengthOfPulse)$. This command directly sets the length of the PWM for the servo, thus allowing for a higher precision. Use the max and min PWM-values from the datasheet for the 0° and 180° positions.

(b) In the setup section, initialize the declared variables, and position the servo at its starting position (0°).

(c) The 2 different character inputs "a" and "b" that you send from the C-code are read through the serial port: One input resets the servo to its starting position, the other input triggers the servo motor to advance one step corresponding to one second. Implement these two behaviours in the loop section.

Hint: Be careful not to add up rounding errors, as **writeMicroseconds()** only works with *integer* values and the angle/pulse length that you calculated will have decimal points and should be in the $float$ format.
Example: Assume the servo step for 1 second is 2.6°. As an int this will be represented as 2°. If you add the next step to the rounded 2° you will get 4.6°, which again, as an int will be rounded to 4°. The actual angle however would be $2 * 2.6^\circ$, or 5.2°, resulting in the more accurate rounded integer value of 5°.

8. Record a video of your working timer in parallel with the console inputs and outputs and send it to your assistants. (**PostLab Q11**)

4.4 Postlab and lab report

1. Upload a single PDF-file with your solution to moodle. The file should contain your answers to the postlab questions as well as the code in the μC ino and μP C files. Please upload your solution in time (before next lab session), late submissions will not be corrected. Please also share the videos for Postlab Q4 and Q11 with your assistants on slack.

2. Come prepared with the Prelab procedure for the next lab.