Simulation of a Quadcopter with Foldable Arms

Martin Ziran Xu, Nathan Bucki

***Abstract*— A novel quadcopter design with foldable arms is presented in this work. By including hinges and springs at the arms, the quadcopter is able to fly through gaps smaller than its actual size, by folding in its arms. In this work the full dynamics of the system is derived, an approach for simulating the system presented and control strategies under various test cases evaluated.**

I. INTRODUCTION

Recent research has demonstrated [1], that quadcopters can be used for search and rescue applications due to their versatile and agile performance. While many work has been done to improve the ability of quadcopters to fly through tiny gaps [2], they only focus on the trajectory generation and are limited by the physical size of the quadcopter. In our work, we present a novel quadcopter concept, which is capable of flying through tiny gaps by folding its arms. In that way, the quadcopter is able to fly through gaps smaller than its actual width. To keep the actual design as simple as possible, we decided to not include any extra actuation at the hinges. Instead, we just use the motor thrust to extend the arms and springs to fold them back in.
In this work, we will focus on the dynamics derivation and simulation. In the last chapter, we will show simulation results, which show the system's behavior under various test cases. Based on that, a first assessment of the idea's feasibility is given.

II. DERIVATION OF DYNAMICS

In this section the derivation of the full vehicle dynamics for arm and quadcopter is presented.

A. Notation

For deriving the dynamics, the tensor notation and style of P.H. Zipfel [3] is used.

- Bold lowercase letters such as $\mathbf{v}$ represent first-order tensors, bold uppercase letters such as $\mathbf{V}$ represent second-order tensors and non-bold symbols such as m represent scalars.
- Upper- and lower-case versions of the same letter will refer to the same quantity.
- A tensor superscript will refer to a frame or a body. ω^{BE} for example denotes the angular velocity of body B wrt. frame E.
- A tensor subscript will refer to a point. $\mathbf{s}_{AB}$ for example denotes the displacement of point A wrt. point B.
- $D^A\mathbf{B}$ refers to the rotational time derivative of tensor quantity $\mathbf{B}$ wrt. frame A.
- A bar over a tensor is used to indicate *transpose*, e.g. $\bar{\mathbf{x}}$
- Square braces, with a superscript, will indicate the expression of a tensor in a coordinate system. $[\mathbf{x}]^B$ for example is tensor $\mathbf{x}$ expressed in the coordinate system B.

The symbols for quantities used in this report are declared in table I.

TABLE I
CLARIFICATION OF SYMBOLS USED IN THIS REPORT

Symbol	Meaning
m^B	mass of body B
$\mathbf{I}_B^B$	inertia tensor of body B with reference point B
$\mathbf{g}$	gravitation
$\mathbf{f}^B$	force acting on body B
$\mathbf{n}_B$	moment at reference point B
$\mathbf{s}_{HB}, \mathbf{S_{HB}}$	displacement of point H wrt. point B
$\mathbf{v}_\mathbf{B}^\mathbf{E}$	translational velocity of point B wrt. frame E
$\omega^{BE}, \mathbf{\Omega}^{BE}$	angular velocity of body B wrt. frame E

B. System

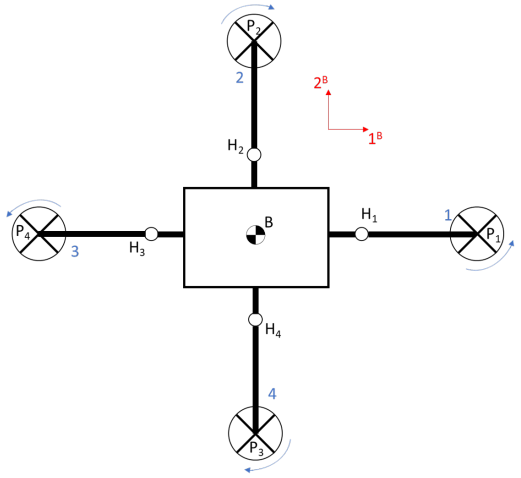

Fig. 1. Simplified model of Quadcopter with foldabe arms seen from a top view

Figure 1 shows a simplified model of our quadcopter system. The main difference to conventional quadcopters results through the introduction of hinges at position $H_1 - H_4$, which allow the arms to fold in during flight time. Note, that no extra actuations has been added. Instead constant force springs are used, which will pull back the quadcopter arms, when the motor thrust is cut. Similar, the arms can

This work was conducted as an undergraduate research project in Spring 2018 in the High Performance Robotics Lab (HiPeRLab) under the supervision of Prof. Mark W. Mueller and Nathan Bucki

be extended through exerting big enough motor thrust. As a result, we are now dealing with an underactuated system with 5 moving and coupled bodies: quadcopter body B and each of the 4 arms A_1, A_2, A_3, A_4.

C. Approach

Dealing with clustered bodies, many approaches can be taken to derive its dynamics. We have decided to devide the quadcopter into each of its moving bodies (quadcopter body + 4 arms) and derive the equations of motion for each of them separately. Note, that we need to introduce reaction forces $\mathbf{f}_{r,i}$ and moments $\mathbf{n}_{r,i}$ at the interconnections. The subscript i indicates the Arm A_i. Figure 2 is displaying them for the body and one arm. According to Newton's third law, reaction forces and moments at the same interconnection are pointing towards opposite directions.
Overall, this approach has the advantage, that we only need to derive the equations of motion for the body and one out of the 4 arms. The other 3 arms will automatically have the same behavior.

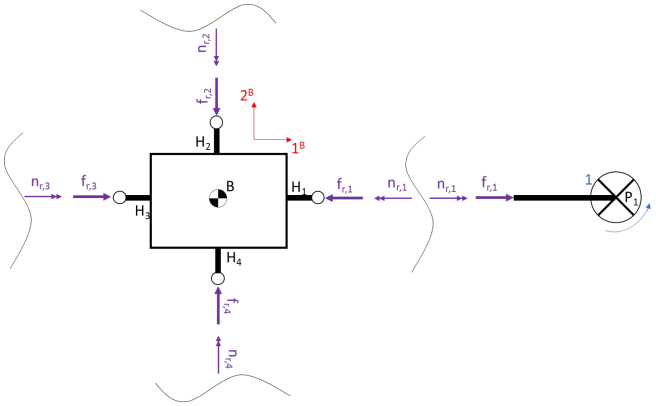

Fig. 2. Reaction forces and moments at interconnections between body and arms

D. Frames and Transformation Matrices

Before deriving the full dynamics, the following frames are introduced (see Figure 3):

- **Earth-fixed-frame** E: acts as the inertia-frame of the system
- **Body-fixed-frame** B: Marks the orientation of the quadcopter body
- **Arm-fixed-frame** A_i: Marks the orientation of the arm A_i. Figure 3 shows arm A_1 folded at angle φ_1. The angle is defined to be $0°$ at full extension and $90°$, when it is fully folded. Note, that wrt. the body the angle φ_1 is the only degree of freedom.
- **Propeller-fixed-frame** P_i: Marks the orientation of the propeller. The 3^{P_i}-axis always aligns with the 3-direction of the arm-fixed frame, but points towards the direction such that the spinning direction of the propeller is positive.
- **Hinge-fixed-frame** H_i: This frame will help us define the transformation matrix $[T]^{BA_i}$. It aligns with the body-fixed frame in the 3-direction, but is rotated, such that 1^{Hi} points to point H_i.

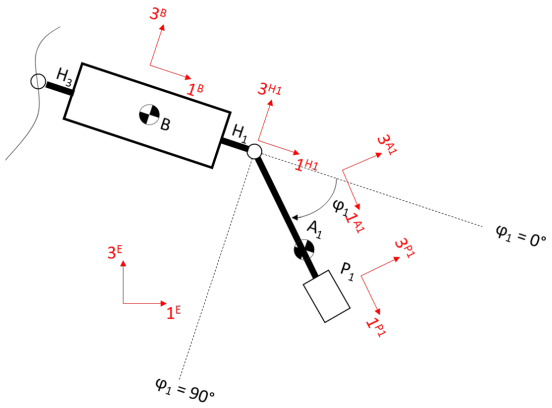

Fig. 3. Sideview of the quadcopter body and one arm showing the different frames

Each of the frames has its preferred coordinate system, in which we will coordinate the tensor quantities. In order to transform a quantity we need the following transformation matrices:

- $[T]^{BE}$: This will be a function of the orientation of the quadcopter body wrt. the earth-fixed frame and can be calculated from the Euler-Symmetric-Parameter presentation of the quadcopter.
- $[T]^{BA_i}$: This can be broken down into $[T]^{BA_i} = [T]^{BH_i} \cdot [T]^{H_i A_i}$. $[T]^{BH_i}$ will be a constant matrix and fully defined through the displacement $s_{H_i B}$. $[T]^{H_i A_i}$ is fully defined by the angle φ_i and will thus vary depending on the arm's orientation.

E. Equation of motions

Euler's law and Newton's law are derived for the quadcopter body and arms. Figure 4 shows the forces and moments acting on both body and arm A_1 in a side view of the system. The reaction forces/moments of the other arms are the same as for arm A_1 and thus not explicitly displayed in the figure. Gravity $\mathbf{g}$ is acting at the center of mass B and A_1 in negative 3^E-direction, and propeller thrust $f_{p,1}$ and torque $n_{p,1}$ in positive 3^A-direction. Furthermore the spring moment $\mathbf{n_s}$ is acting on both arm and body in the 2^A-direction.

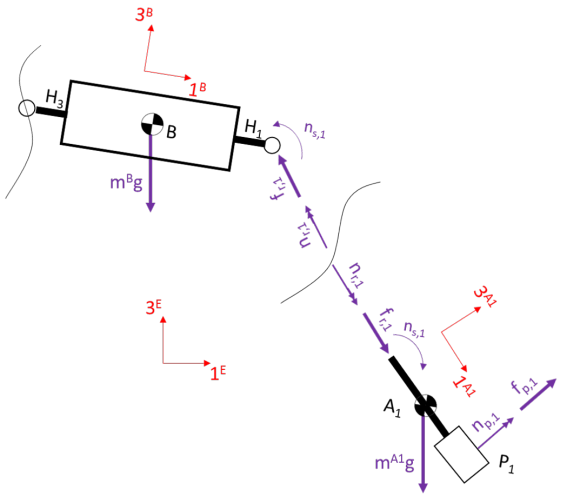

Fig. 4. Forces acting on quadcopter body and arm A_1

1) Body: Equation of motions

Euler's Law:

$$D^E(\mathbf{I}_B^B \omega^{BE}) = \sum \mathbf{n}_B \tag{1}$$

$$\mathbf{I}_B^B D^B \omega^{BE} + \mathbf{\Omega}^{BE} \mathbf{I}_B^B \omega^{BE} = \sum_{i=1}^{4} (-\mathbf{n}_{r,i} - \mathbf{S}_{HB}\mathbf{f}_{\mathbf{r},\mathbf{i}} - \mathbf{n}_{\mathbf{s},\mathbf{i}}) \tag{2}$$

Newton's Law:

$$D^E(m^B \mathbf{v}_B^E) = \sum \mathbf{f}^B \tag{3}$$

$$m^B D^E(\mathbf{v}_B^E) = \sum_{i=1}^{4} -\mathbf{f}_{\mathbf{r},\mathbf{i}} + m^B \mathbf{g} \tag{4}$$

Note, that the quantities $\mathbf{f}_{r,i}, \mathbf{n}_{r,i}, \mathbf{n}_{s,i}$ are counting towards equation (2),(4) as negative quantities. It is just a convention, but it will become important, when we write down the arm's equations of motion. Those quantities will then, due to Newton's third low, count as positive ones.

2) Arm 1: Equation of motions

As mentioned above, the derivation will be the same for all arms and can thus be shown exemplary on arm 1. In the sake for simplicity A will be used for the arm-fixed frame instead of A_1.

Euler's Law:

$$D^E(\mathbf{I}_A^A \omega^{AE} + \mathbf{I}_P^P \omega^{PE}) = \sum \mathbf{n}_A + \sum \mathbf{n}_P \tag{5}$$

Note, that we are including the propeller dynamics into the system as well. After transforming equation (5) into the correct frames, it yields:

$$\begin{aligned} &(\mathbf{I}_A^A + \mathbf{I}_P^P)(D^A\omega^{AB} + D^B\omega^{BE}) \\ &+\mathbf{I}_P^P D^P \mathbf{w}^{PA} + \mathbf{I}_P^P((\mathbf{\Omega}^{BA} + \mathbf{\Omega}^{AP})\omega^{BE} + \mathbf{\Omega}^{AP})\omega^{AB} \\ &+\mathbf{I}_A^A \mathbf{\Omega}^{BA}\omega^{BE} + (\mathbf{\Omega}^{AB} + \mathbf{\Omega}^{BE})\mathbf{I}_A^A(\omega^{AB} + \omega^{BE}) \\ &+(\mathbf{\Omega}^{PA} + \mathbf{\Omega}^{AB} + \mathbf{\Omega}^{BE})\mathbf{I}_A^A(\omega^{PA} + \omega^{AB} + \omega^{BE}) \\ &= \\ &\mathbf{n}_{\mathbf{r},\mathbf{1}} + \mathbf{S}_{H_1A_1}\mathbf{f}_{\mathbf{r},\mathbf{1}} \\ &+\mathbf{n}_{\mathbf{p},\mathbf{1}} + \mathbf{S}_{P_1A_1}\mathbf{f}_{\mathbf{p},\mathbf{1}} + \mathbf{n}_{\mathbf{s},\mathbf{1}} \end{aligned} \tag{6}$$

Newton's Law:

$$D^E(m^A \mathbf{v}_A^E) = \sum \mathbf{f}^A \tag{7}$$

Transforming equation (7) into desirable frames, gives us:

$$\begin{aligned} &m^A[D^E\mathbf{v}_B^E - \mathbf{S}_{AH}D^A\omega^{AB} - (\mathbf{s}_{AH} + \mathbf{s}_{HB})D^B\omega^{BE} \\ &(\mathbf{\Omega}^{BE}\mathbf{\Omega}^{BE} + (\mathbf{\Omega}^{AB} + \mathbf{\Omega}^{BE})\mathbf{\Omega}^{AB} + \mathbf{\Omega}^{BE}\mathbf{\Omega}^{AB})\mathbf{s}_{AH} \\ &+\mathbf{\Omega}^{BE}\mathbf{\Omega}^{BE}\mathbf{s}_{HB}] \\ &= \\ &\mathbf{f}_{\mathbf{p},\mathbf{1}} + \mathbf{f}_{\mathbf{r},\mathbf{1}} + m^A\mathbf{g} \end{aligned} \tag{8}$$

Equations (2), (4) for the body and (6), (8) for each of the arms denote 10 tensor equations and fully describe the dynamics of the system.

III. Simulation Approach

As a result of last section, we have derived 10 tensor equations for describing the dynamics of the system. If we coordinate those equations in a preferred coordinate system, we have a set of 30 coupled, nonlinear differential equations to solve for the attitude and translational dynamics of the body and each of the 4 arms.

As a common strategy, we can approximate the solution with a one-step Euler approximation. By doing so, we assume that all velocities, attitude and position terms are constant over a time step. We can then solve for the acceleration terms $D^B\omega^{BE}$, $D^A\omega^{AB}$, $D^E\mathbf{v}_B^E$ at each time step and calculate the velocity, attitude and position of the next time step through Euler integration.

A. Overview of quantities

Before presenting a method to solve for the dynamics, this section should give an overview over the tensor quantities in the dynamics equations (2), (4), (6), (8). They will be divided into 3 classes: known constants, known at each time step and unknowns (to solve for). The subscript i stands for the specific arm A_i and goes from $1-4$.

Known Constants:

Mass/inertia:

$$m^B, m^{A_i} \tag{9}$$

$$[\mathbf{I}_B^B]^B = diag(I_1^B, I_2^B, I_3^B) \tag{10}$$

$$[\mathbf{I}_{A_i}^{A_i}]^{A_i} = diag(I_1^{A_i}, I_2^{A_i}, I_3^{A_i}) \tag{11}$$

Displacements:

$$[\mathbf{s}_{A_iH_i}]^{A_i} = (l_{A_iH_i}, 0, 0)' \quad (12)$$
$$[\mathbf{s}_{P_iH_i}]^{A_i} = (l_{P_iH_i}, 0, 0)' \quad (13)$$
$$[\mathbf{s}_{H_iB}]^{B} = (l_{H_iB}, 0, 0)' \quad (14)$$

Gravity:

$$[\mathbf{g}]^{E} = (0, 0, -9.81)' \quad (15)$$

Constant at each time step:
Angular velocities:

$$[\omega^{BE}]^{B} = (p, q, r)' \quad (16)$$
$$[\omega^{A_iB}]^{A_i} = (0, \dot{\varphi}_{A_i}, 0)' \quad (17)$$
$$[\omega^{P_iA_i}]^{A_i} = (0, 0, (-1)^{i+1}\omega_{p_i})' \quad (18)$$
$$[D^{P}\omega^{P_iA_i}]^{A_i} = (0, 0, (-1)^{i+1}\dot{\omega_{p_i}})' \quad (19)$$

In equation (17) we enforced the constraint, that wrt. the body-frame the arm-frame can only rotate in 2^A-direction. Similar in equations (18) and (19), where wrt. the arm-frame the propeller can only rotate in the 3^A-direction. The $(-1)^{i+1}$ denotes the different spinning direction of the propellers. The propeller speed $\omega_{p,i}$ can be treated as a known input to the system.
Forces and moments:

$$[\mathbf{f}_{P_i}]^{A_i} = (0, 0, \kappa_1\omega_{p_i}^2)' \quad (20)$$
$$[\mathbf{n}_{\mathbf{P_i}}]^{A_i} = (0, 0, (-1)^{i}\kappa_2\omega_{p_i}^2)' \quad (21)$$
$$[\mathbf{n}_{\mathbf{s,i}}]^{A_i} = fun([T]^{BA_i}) \quad (22)$$

Equation (20) and (21) show the relationship between propeller speed w_{p_i} and the thrust and torque generated. κ_1, κ_2 are constants. Note, that the drag torque is always acting in the opposite of the spinning direction. Calculating the spring moment $n_{s,i}$ is not as straight forward and will be a function of the orientation of the arm A_i wrt. the body B. It will be explained in more detail in the implementation section (IV. A).
Lastly, the orientations $[T]^{BE}$ and $[T]^{AB}$ (as defined in the last section) can also be assumed to be constant over a time step. Inside the simulation they are processed using Euler Symmetric Parameters.

Unknowns:
Angular/translational accelerations:

$$[D^{B}\mathbf{v}_B^E]^{E} = (a_1, a_2, a_3)' \quad (23)$$
$$[D^{B}\omega^{BE}]^{B} = (\dot{p}, \dot{q}, \dot{r})' \quad (24)$$
$$[D^{A}\omega^{A_iB}]^{A_i} = (0, \ddot{\varphi_{A_i}}, 0)' \quad (25)$$

Reaction forces and moments:

$$[\mathbf{f}_{r,i}]^{A_i} = (f_{1,i}, f_{2,i}, f_{3,i})' \quad (26)$$
$$[\mathbf{n}_{r,i}]^{A_i} = (n_{1,i}, n_{2,i}, n_{3,i})' \quad (27)$$

Equations (23)-(27) denote the quantities to solve for. While we are not directly interested in the reaction forces and moments, they still need to be considered as unknowns.
By using the one-step Euler approximation we have transformed the problem of solving a nonlinear, coupled set of differential equations into a problem of solving for unknown quantities. Figure 5 gives an overview of those quantities.

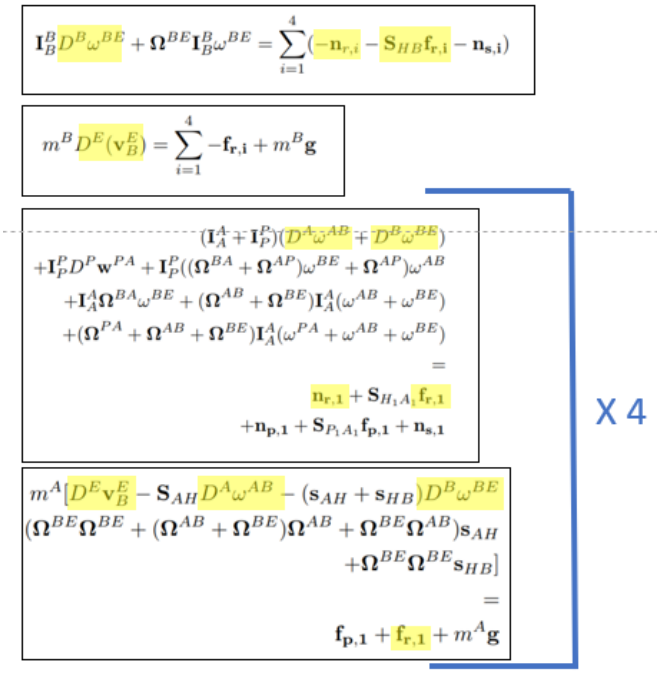

Fig. 5. Derived equation of motions with highlighted parts being the unknown quantities to solve for.

B. Solving for unknowns

Since the system dynamics are linear in the unknown quantities, we can solve them using a matrix approach. We coordinate equations (2), (4), (6), (8) in the B-CS and reorder those equations, such that all unknown quantities are on the left and all known quantities are on the right side of the equation. We abbreviate the constant right side of the equations as $[C_i]^B$. This gives us:

$$[\mathbf{I}_B^B]^B[D^B\omega^{BE}]^B + \sum_{i=1}^{4}([T]^{BA_i}[\mathbf{n}_{\mathbf{r,i}}]^{A_i} + [\mathbf{S}_{HB}]^B[T]^{BA_i}[\mathbf{f}_{\mathbf{r,i}}]^{A_i}) = [\mathbf{C_1}]^B \quad (28)$$

$$m^B[T]^{BE}[D^E\mathbf{v}_B^E]^E + \sum_{i=1}^{4}[T]^{BA_i}[\mathbf{f}_{\mathbf{r,i}}]^{A_i} = [\mathbf{C_2}]^B \quad (29)$$

$$[\mathbf{I}_A^A + \mathbf{I}_P^P]^B([T]^{BA}[D^A\omega^{AB}]^A + [D^B\omega^{BE}]^B) - [T]^{BA}[\mathbf{n}_{\mathbf{r},1}]^A - [\mathbf{S}_{HA}]^B[T]^{BA}[\mathbf{f}_{\mathbf{r},1}]^A = [\mathbf{C_{3,1}}]^B \quad (30)$$

$$m^A([T]^{BE}[D^E\mathbf{v}_B^E]^E - [\mathbf{S}_{AH}]^B[T]^{BA}[D^A\omega^{\mathbf{AB}}]^A \\ -([\mathbf{s}_{AH}+\mathbf{s}_{HB}]^B)[D^B\omega^{BE}]^B) - [T]^{BA}[\mathbf{f}_{\mathbf{r},\mathbf{1}}]^A \\ = [\mathbf{C}_{\mathbf{4},\mathbf{1}}]^B \quad (31)$$

The matrix approach will only be shown in detail for a system with one arm and one body. Once understood, the system can be easily scaled up to 4 arms. First, all the unknown quantities are stacked into a 15x1 solution vector x.

$$x = [[D^B\omega^{BE}]^B, [D^E\mathbf{v}_B^E]^E, [D^A\omega^{AB}]^A, [\mathbf{n}_{\mathbf{r},\mathbf{1}}]^A, [\mathbf{f}_{\mathbf{r},\mathbf{1}}]^A]' \\ = [\dot{p},\dot{q},\dot{r},a_1,a_2,a_3,0,\dot{\varphi}_1,0,n_1,n_2,n_3,f_1,f_2,f_3]' \quad (32)$$

Similar all constants on the right side of equations (28)-(31) can be stacked into a 12x1 vector b.

$$b = [[C_1]^B, [C_2]^B, [C_3]^B, [C_4]^B]' \quad (33)$$

The set of equations can then be rewritten as:

$$Ax = b \quad (34)$$

$$\begin{bmatrix} [\mathbf{I}_B^B]^B & 0 & 0 & [T]^{BA} & [\mathbf{S}_{HB}]^B[T]^{BA} \\ 0 & m^B[T]^{BE} & 0 & 0 & [T]^{BA} \\ [\mathbf{I}_A^A+\mathbf{I}_P^P]^B & 0 & [\mathbf{I}_A^A+\mathbf{I}_P^P]^B[T]^{BA} & -[T]^{BA} & -[\mathbf{S}_{HA}]^B[T]^{BA} \\ -m^A([\mathbf{s}_{AH}+\mathbf{s}_{HB}]^B) & m^A([T]^{BE} & -m^A[\mathbf{S}_{AH}]^B[T]^{BA} & 0 & -[T]^{BA} \end{bmatrix} \begin{bmatrix} [D^B\omega^{BE}]^B \\ [D^E\mathbf{v}_B^E]^E \\ [D^A\omega^{AB}]^A \\ [\mathbf{n}_{\mathbf{r},\mathbf{1}}]^A \\ [\mathbf{f}_{\mathbf{r},\mathbf{1}}]^A \end{bmatrix} = \begin{bmatrix} [\mathbf{C_1}]^B \\ [\mathbf{C_2}]^B \\ [\mathbf{C_{3,1}}]^B \\ [\mathbf{C_{4,1}}]^B \end{bmatrix}$$

Fig. 6. Rewriting the tensor equations into matrix vector form for body and one arm

Note, that the A-matrix has dimension 12x15. In order to solve the system, we need to enforce constraints and delete all columns of A, which matches a zero entry in the solution vector x. Equation (32) shows, that $x(7)$ and $x(9)$ are already zero, due to the constraint on the quadcopter arm. The third and last constraint depends on the state of the arm, i.e. there are 2 cases. If the arm is not fully stretched (case 1), $\ddot{\varphi}$ is not zero, but the second component of the reaction moment at the Hinge $x(11) = n_2$ will be zero, since the hinge allows the arm to move freely in that direction. In that case, we can delete columns 7,9,11 of the A matrix and get rid of rows 7,9,11 of the solution vector x. We define that set of equations to be:

$$\hat{A}\hat{x} = b \quad (35)$$

$\hat{A}$ is now of dimension 12x12 and invertable. $\hat{x}$ contains all our unknown quantities and can now be calculated through:

$$\hat{x} = \hat{A}^{-1}b \quad (36)$$

The procedure is similar to case 2, where the arms are fully stretched and not moving. $\ddot{\varphi}$ will then be zero, while n_2 is not. In that case column 7,8,9 of the A matrix and row 7,8,9 of the solution vector x should be deleted to build $\hat{A}$ and $\hat{x}$. Note, that the reduced solution vector $\hat{x}$ needs to be interpreted in a different way depending on the state of the arm.

C. Euler Integration

Having solved for $\hat{x}$ we now have the angular acceleration for the body $\dot{\omega}^{BE}(k)$, the angular acceleration of the arm $\ddot{\varphi}(k)$ as well as the translational acceleration of the body $a_B^E(k)$ at a specific time step k. Explicitly they can be extracted as follows:

$$\dot{\omega}^{BE}(k) := (\dot{p},\dot{q},\dot{r})' = [\hat{x}(1),\hat{x}(2),\hat{x}(3)]' \quad (37)$$

$$a_B^E(k) := (a_1,a_2,a_3)' = [\hat{x}(4),\hat{x}(5),\hat{x}(6)]' \quad (38)$$

In terms of the angular acceleration of the arm, we need to split into two cases. For case 2 (arms being fully stretched) $\ddot{\varphi}(k) = 0$ by definition. For case 1 it applies:

$$\ddot{\varphi}(k) = \hat{x}(7) \quad (39)$$

We will see later, how to switch between those two cases in the code implementation section.
Using those accelerations, the whole state of the system can be constructed using Euler integration with a defined time step dt. Let ω^{BE}, v_B^E, and s_{BE} be the angular velocity, translational velocity and position of the body respectively:

$$v_B^E(k+1) = v_B^E(k) + dt \cdot a_B^E(k) \quad (40)$$

$$s_{BE}(k+1) = s_{BE}(k) + dt \cdot v_B^E(k) \quad (41)$$

$$\omega^{BE}(k+1) = \omega^{BE}(k) + dt \cdot \dot{\omega}^{BE}(k) \quad (42)$$

For the attitude let $att^{BE}(k)$ stand for the current attitude of the quadcopter. $att_{change}(k)$ denotes the attitude change within the time step dt and can be calculated from the current angular velocity $\omega^{BE}(k)$ and time step dt. The new attitude $att^{BE}(k+1)$ can then be calculated as the composition of $att^{BE}(k)$ and $att_{change}(k)$.

$$att_{change}(k) = from\_rotation\_vector(\omega^{BE}(k)dt) \quad (43)$$

$$att^{BE}(k+1) = compose(att^{BE}(k), att_{change}(k)) \quad (44)$$

Finally, the Euler Integration is very straightforward for the arm and given through:

$$\dot{\varphi}(k+1) = \dot{\varphi}(k) + dt \cdot \ddot{\varphi}(k) \quad (45)$$

$$\varphi(k+1) = \varphi(k) + dt \cdot \dot{\varphi}(k) \quad (46)$$

After evaluating equations (40)-(46), all current velocities and orientations are known. One can thus repeat the procedure and solve for the accelerations of the next time step, using the matrix approach, presented in the previous section.

D. Scaling:

Although derived for only one arm, this procedure can be easily scaled for the whole system. Each of the arms will introduce 2 new tensor equations and 6 new unknowns (reaction forces, moments and $\ddot{\varphi}$). For the whole system, matrix A and x will then be of dimension 30x42 and 42x1. After enforcing constraints, we will be left over with $\hat{A}$ and $\hat{x}$ of dimension 30x30 and 30x1, which can be solved by inverting $\hat{A}$.

IV. Implementation

Using the approach, derived in the last section, we can implement the simulation as follows:

1) Initialize system at time k = 0 with initial state.
2) Set propeller speed $\omega_{p,i}$ for each of the motors.
3) Solve for unknown accelerations, reaction forces and moments
   - Calculate constant right side terms $[C_i]^B$ and A matrix using velocities, orientations and external forces (propeller thrust/torque, spring moment) from current time step. Build up b vector through $[C_i]^B$.
   - Enforce constraints and solve for solution vector $\hat{x}$ through $\hat{x} = \hat{A}^{-1}b$.
   - Assign accelerations and reaction forces/moments from $\hat{x}$.
4) Calculate velocities, positions and orientations of the next time step through Euler integration.
5) Check, in which case the arms are: stretched or folded
6) Repeat from step 2.

A. Detailed models:

In this section, we will go into a few details of the implementation and refer to specific steps of the algorithm presented in the last part.

1) Motor model: The angular velocities of the propellers $\omega_{p,i}$ are set as an input to the system in Step 2) of the simulation. Since changes in the angular velocities cannot happen arbitrarily fast in real life, we have also included a time constant τ_m in the simulation. Let $\omega_{des}(k)$ be the desired angular velocity of the propeller. The propeller speed of the next time step $\omega_p(k+1)$, is then calculated as:

$$\omega_p(k+1) = c\omega_p(k) + (1-c)\omega_{des}(k) \tag{47}$$

with:

$$c = exp(-dt/\tau_m) \tag{48}$$

The angular acceleration of the propeller is then calculated as:

$$\dot{\omega_p}(k) = \frac{\omega_p(k) - \omega_p(k-1)}{dt} \tag{49}$$

2) Spring model: We have decided to use constant force springs in our physical model for pulling back the arms. Figure 7 shows an illustration of the spring. It is mounted between body and arm at point S_1 and M_1. We assume, that the scalar value of the spring force f_s is constant. However, the resulting spring moment at the hinge H_1 will be different depending on the orientation of the arm wrt. the body. We will introduce a spring frame S, which aligns with the arm-frame in the 2-direction, but always points in the direction of the spring in 1-direction.
Figure 8 shows the spring force $\mathbf{f_s}$ acting on the arm. We are interested in the moment exerted by $\mathbf{f_s}$ at point H and define it to be the spring moment n_s. It can be calculated as follows:

$$\mathbf{n_s} = \mathbf{s}_{MH} \times \mathbf{f_s} \tag{50}$$

Since $[\mathbf{f_s}]^S = (-f_s, 0, 0)'$ is known, we need to find the transformation matrix $[T]^{AS}$, which can be calculated through $\mathbf{s}_{MS}$ as follows:

$$[\mathbf{s}_{MS}]^A = [\mathbf{s}_{MH}]^A - [T]^{AB}[\mathbf{s}_{SH}]^B =: (s_1, s_2, s_3)' \tag{51}$$

$[T]^{AS}$ can then be calculated through the rotation around the negative 2^A-direction by angle $\gamma = atan2(s_3, s_1)$

$$[T]^{AS} = rotationVectorToMatrix((0, -1, 0)' \cdot \gamma) \tag{52}$$

The spring moment can then be calculated through:

$$[\mathbf{n_s}]^A = [\mathbf{s_{MH}}]^A \times ([T]^{AS}[\mathbf{f_s}]^S) \tag{53}$$

In order to find the optimal position of the mounting point S ($\mathbf{s}_{SH}$) and M ($\mathbf{s}_{MH}$), we have optimized over the time, it takes to fold in and stretch.

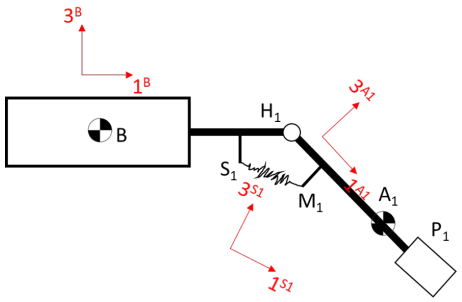

Fig. 7. Illustrating constant force spring mounted between point S_1 and M_1 at the body and arm respectively.

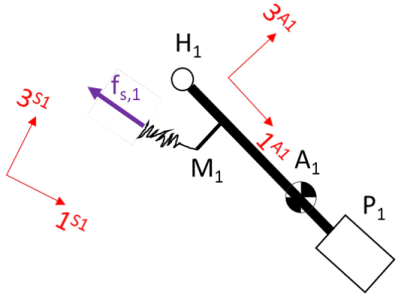

Fig. 8. Showing spring force $\mathbf{f_s}$ acting on the arm.

3) State of the arm: In section III, we have seen, that it is crucial to know, if the arm is folded or stretched, since the constraints will depend on that. That is why step 5) of the simulation explicitly requests to check the state of the arm. For that purpose, we introduce a boolean variable $folded$ for each of the arms, which will be true, if $\varphi = (0°, 90°]$. For switching between $True$ and $False$ the following algorithm is implemented. It will be shown in pseudo code and exemplary for one arm.

```
if (folded==true):
        phi_ddot = x_hat(7)
        updateArm()
    if (phi<0):
        phi = 0
        phi_dot, phi_ddot = 0
        folded = false
    if (phi>pi/2):
        phi = pi/2
        phi_dot, phi_ddot = 0
if(folded == false):
        n_2 = x_hat(8)
    if (n_2<0):
        n_2 = 0
        folded = true
```

When the arm is moving in the folded state, we are retrieving the angular acceleration $\ddot{\varphi}$ and update the angle and its velocity at each time step. If the arm hits one of the boundaries, we set the velocity and acceleration to zero and change the state $folded$ to $false$, if it has reached $0°$.
If $folded = false$ we are retrieving the reaction force in 2^A direction instead. Nothing needs to be updated, since we know, that the the angular acceleration is zero. However, we need to be careful, if the arm wants to fold back in. Coordinated in the A-CS the second component of the reaction moment n_2 can only act in positive direction, preventing the arm to go further up. If $n\_2 < 0$, we know that the arm wants to fold back in and we can set $folded$ back to $true$.
Note, that we do not calculate the reaction moment n_2, when the arm hits the $90°$ boundary. Since the motor thrust can only act in one direction, n_2 is only acting in the moment of the impact.

V. Control Strategy:

Same as for a normal quadcopter, the propeller speed of each of the motors is the only input, we can manipulate. In the following section, the "normal-flight" scenario and "fly-through-gap" scenario is presented. The control strategies have been tested using the simulation explained in the first part of the report. If not otherwise specified, the parameters of table II have been used in the simulation. Note, that all values are given in SI-dimensions.

A. Normal Flight

For normal flight our control strategy is using the most simple approach possible. For extending the arms, we just set all motor thrust to the maximum value. After all arms are fully extended, the system basically behaves like a conventional quadcopter. We can thus use the standard quadcopter controller for attitude and position control. The only difference is, that we need to make sure, that the motor thrust never goes under a specific lower bound, which is defined through the spring force. Figure 9 shows all forces acting on the arm in the extended state. To prevent the arm from folding back in, we need to enforce the condition, that the reaction torque in 2^A-direction is always greater than zero. Since the arm is not moving during that time, we can simply take the static torque balance around point H_1 to determine the minimum thrust needed:

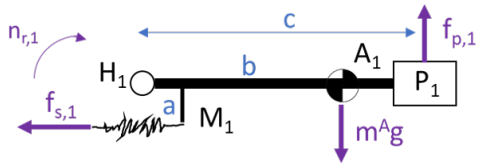

Fig. 9. Illustrating the forces acting on the arm during full extension

$$n_{r,1} = f_{p,1}c - f_{s,1}a - m^A gb > 0 \tag{54}$$

$$f_{p,1} > \frac{f_{s,1}a + m^A gb}{c} \tag{55}$$

As long as we guarantee that lower constraint, the arms should stay extended. Finally, if we want to fold back the arms, we just cut off the motor thrust to be zero and the spring will pull them back in.

This control strategy is tested in **TestRun1**. Figure 10 shows the angle φ_1 of Arm 1 in the first subfigure, the reaction torque at the hinge in 2^A-direction in the second, the absolute value of the spring torque in the third and the motor thrust in the fourth. During the first 0.2 seconds we see the arms unfolding from 90 degrees to 0 degrees with maximum motor thrust. Note, that the hinge moment is zero and the spring moment some nonlinear function of the angle. After 0.2 seconds the normal quadcopter controller is activated. We can see in figure 11, that the position of the quadcopter is roughly stabilized after 2 seconds. Finally, after 2.5 seconds the motors are turned off, which causes the arms to fold back and the position to drop.

TABLE II
Parameters for simulation (given in SI-dimensions)

Quantity	value
dt	0.01
m^B	0.27
I_B^B	diag(2.5e-4, 2.5e-4, 2.5e-4)
m^A	0.098
I_A^A	diag(1.4e-5, 2.7e-4, 2.7e-4)
dist_ArmHinge	0.1
dist_PropHinge	0.138
dist_HingeBody	0.028
κ_1	6.4e-6
κ_2	1.1e-7
I_P^P	diag(0, 0, 15e-6)
τ_m	0
motMaxSpeed	800
f_s	2
$[s_{SH}]^{A_1}$	[0.02, 0, 0.04]'
$[s_{MH}]^{A_1}$	[0.1, 0, -0.02]'
posCtrlNatFreq	2
posCtrlDampingRatio	0.7
timeConstAngleRP	0.2
timeConstAngleY	1
timeConstRatesRP	0.03
timeConstRatesY	0.5

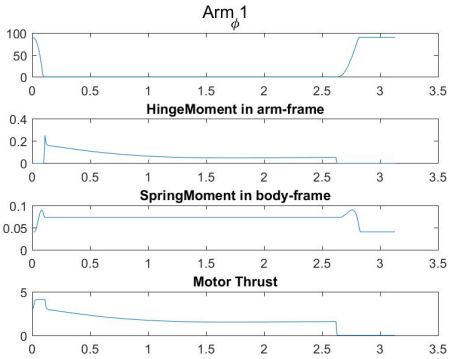

Fig. 10. Plots showing the angle, reaction torque in 2^A-direction, absolute value of the spring torque and motor thrust for arm 1 in **TestCase1**: unfolding, hover control and folding

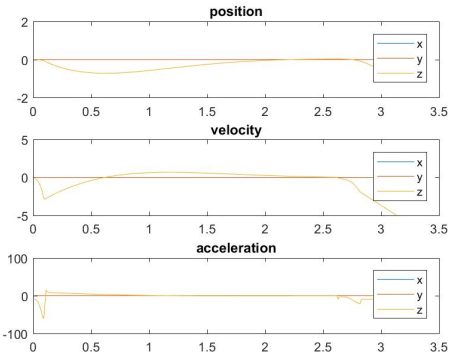

Fig. 11. Plots showing the translational dynamics of the body in **TestCase1**: unfolding, hover control and folding

B. Fly through gap

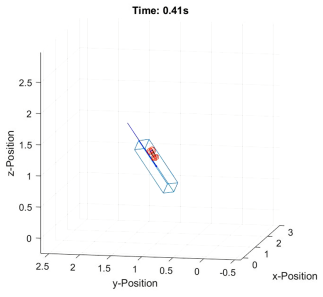

Fig. 12. Showing a screenshot of the animation displaying the quadcopter inside the gap with its arms folded

The challenge of flying through a gap is, that once the motor arms are folded, no control actions can be made. Our approach to generate the trajectory is to treat the quadcopter as a point mass and project its motion through the gap. We can thus solve for an initial velocity and position, the quadcopter has to reach, when closing its arms, which will result in the body flying through that gap. In addition to that, we can also give an estimate of how long the quadcopter will be flying.

We can then use a trajectory generation algorithm [4] to reach that desired velocity and position and close our arms for the specified time. Afterwards, the arms are stretched and the quadcopter is stabilized using the conventional quadcopter controller.

The gap is defined as a cuboid through its length l, height h, width w, center position s_{GE} and its orientation z_G. For the example, presented in the following, we have used the following specifications:

$$l = 1, h = 0.2, w = 0.2 \tag{56}$$
$$s_{GE} = [1, 1, 1]' \tag{57}$$
$$z_G = [1, 1, 1]' \tag{58}$$

In order to determine the initial velocity, position and flying time of the quadcopter, we have minimized over the initial velocity. The solver then came up with the following solution:

$$posDes = [0.3660, 0.3660, -0.0957]' \tag{59}$$
$$velDes = [1.6944, 1.6944, 5.0801]' \tag{60}$$
$$T_f = 0.5686s \tag{61}$$

Figure 12 is showing a screenshot of our animation, which displays the quadcopter inside the gap with its arms folded. Figure 13 and 14 are showing the actual results from the simulation (**TestRun2**). We have started with the quadcopter at the desired initial position and velocity. We have furthermore set the attitude of the quadcopter, such that the velocity is pointing into thrust direction. Looking at figure 14, the motor thrust is set to zero at the beginning to allow the arms to fold back. After roughly 0.5686s, which is the time specified through the optimizer, we open up the arms through full thrust and stabilize the system using the conventional controller. Figure 13 shows the position of the body during the test run. We can see, that the system is actually able to fly through the gap[1].

[1]Animated videos are available in the Appendix folder

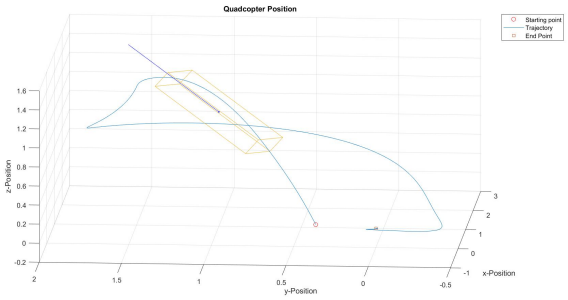

Fig. 13. 3D plot of the body's position in **TestCase2**: Flying through gap without disturbances

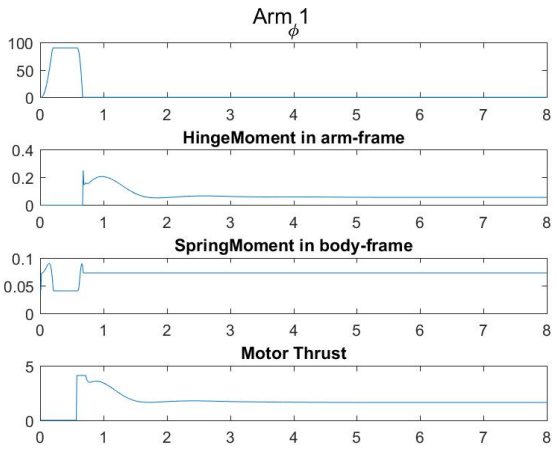

Fig. 14. Plots showing the angle, reaction torque in 2^A-direction, absolute value of the spring torque and motor thrust for arm 1 in **TestCase2**: Flying through gap without disturbances

VI. Various Test Cases

Section V shows the normal flight behavior and gap maneuver under ideal conditions. In the following section we are interested in, how the system behaves under various disturbance cases. Plots and videos to the test cases can be found in the submitted Appendix folder.

A. Asymmetric unfolding

Here, the performance in the unfolding and hovering process under asymmetric disturbances has been examined. We have modeled the disturbance in two different ways.
In the first case (**TestRun3**), the maximum motor speed of arm 1 is set to be 25% lower than the rest of the arms. This will cause arm 1 to produce less thrust in the unfolding process, resulting in some asymmetric behavior of the whole system. Although this induces roll and pitch angles of up to 50° into the system, the quadcopter can still recover to hovering after a few seconds.
In the second case, we have tested the system for different motor time constants. While τ_m is set to be 0.005 for arm 2-4, arm 1 has a motor constant being twice as big (0.01) (**TestRun4**). As a result we see arm 1 unfolding slower, which will again cause some attitude disturbance in the system. Roll and pitch have peak values of up to 20°. Similar to case 1, the quadcopter is able to recover after a few seconds. If we increase the motor time constant of arm 1 to be 5 times bigger (0.05) (**TestRun5**), the disturbance will cause one arm to fold in a little bit, but the controller is still able to stabilize the system at the end.

B. Testing minimum thrust bound for controller

As mentioned in the last section, we have to include a lower bound for the motor thrust in normal flight mode to prevent the springs to pull back the arms. In **TestRun6** we have increased the spring force to be 50% bigger. This will cause the minimum thrust to increase just right below the thrust necessary for hovering.

$$minMotThrust = 1.4829N \tag{62}$$
$$Thrust\_hovering = 1.6236N \tag{63}$$

Our simulation show, that the arms are not folding back during normal flight mode.

C. Angular disturbances in "Flying-through-gap" maneuver

Since no control actions can be taken, once the arms are closed, we are very sensitive to any angular disturbances at the moment of folding in. We use the same setting as for TestRun2, but increase, similar to TestRun4, the motor time constant of arm 1. Given the short flight time of 0.6s, only a very high motor time constant deviation (10 times higher)(**TestRun7**) will cause any noticable spinning behavior. However, the quadcopter already fails to stabilize after the fly through, for the motor time constant being 2 times higher (**TestRun8**). The reason for failure lies, according to the opinion of the author, in the compromised controller behavior due to the motor time constant deviation and not in the angular disturbance. In **TestRun9** we have tested the same system with a different control action. To balance out the higher motor time constant of arm 1, we have set arm 3 to have the same motor time constant after the fly through. The resulting controller is now able to stabilize the system. This opens the possibility of implementing a learning algorithm, to compensate those disturbances.
In another case (**TestRun10**) we have explicitly initialized the system with an angular velocity of $q = 2rad/s$. This causes the quadcopter to spin during fly through, ending up with a pitch and roll peak of almost 180°. Given those extreme conditions, the quadcopter is still able to stabilize itself after extending its arms.

VII. Conclusion

During the period of one semester, we have derived and implemented a simulation of the system. We were able to test out our basic control strategy under various disturbance cases and verify under which circumstances the "flying-through-gap" maneuver is feasible. Overall, we could observe promising results, which convinced us to move forward with that idea.

In the next steps, we will implement the simulation in the "lab-code" to test out a few real-life disturbances, which are modeled in that code base. We will also move forward with the actual physical design to test out our control strategy on real hardware. Finally, we will start thinking about more sophisticated trajectory generation algorithm for the "flying-through-gap" maneuver, which may also include some kind of learning algorithm to compensate disturbances.

APPENDIX

Within the submission of this report, a zip-file containing plots and videos of all the test cases are provided. Furthermore, the Matlab-implementation of the simulation is pushed to the collobarative GitHub folder with Nathan Bucki.

References

[1] S. Waharte and N. Trigoni, "Supporting search and rescue operations with UAVs," in *2010 International Conference on Emerging Security Technologies (EST)*. IEEE, 2010, pp. 142–147.

[2] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza, "Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 5774–5781.

[3] P. H. Zipfel, *Modeling and Simulation of Aerospace Vehicle Dynamics*, 2nd ed. American Institute of Aeronautics and Astronautics, 2007.

[4] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov 2013, pp. 3480–3486.